UNIVERSITETI SHTETEROR I TETOVES

Hap pas hapi C++

Hapi 1

Programerja e parë Agusta Ada Byron

Blerim Sherifi

Tetovë, 2012

ME EMRIN E ALL-LLAHUT MËSHIRUESIT
MËSHIRËBËRËSIT

****

****

****

***

*

Parathënie

Gjuha programuese është një vegël informatike e përpiluar nga operacionet matematikore që shërben për të shkruar domene të cilat përcillen në veglat (makinat vizuale konvertuese) për të fituar kode në gjuhën e makinës. Përndryshe gjuha programuese është një gjuhë e shkruar e cila nuk flitet dhe që përmban një bashkësi të rregullave e që shërben si vegël për komunikimin në mes të programorëve, zhvilluesve dhe makinave inteligjente (zakonisht kompjuteri) apo gjësendi që është në gjendje të kuptojë atë gjuhë. Me ndihmën e tyre përpilohen vegla dhe programe informatike më të zhvilluara që ruhen nga veglat elektronike të dirigjuara nga mikroprocesorët.

Për gjuhët programuese flitet në rastet kur përdoret një bashkësi e përpiluar nga disa fjalë-urdhëresa e komanda (zakonisht me prejardhje nga anglishtja) dhe disa rregulla të përdorimit të tyre. Me ndihmën e kësaj bashkësie përgatitet një program sipas të cilit vepron makina, sistemi operues i makinave inteligjente apo gjësend tjetër që ka aftësi të lexoj programin e përgatitur.

Hyrje

Në fillim të ndjekjes së mësimeve të një gjuhe të huajë njeriu ballafaqohet me fjalë e tinguj që nuk mund ti kuptojë e lëre më të i shpjegojë. Për këtë ai i pranon disa nga fjalët të cilat zakonisht 'bien në vesh' më shpesh dhe në bazë të tyre vazhdon nxënien e fjalëve të reja. Si do që të jetë, në ditët tona për mësimin e gjuhëve të huaja, përdorim libra nga të cilat nxjerrim këshilla dhe udhëzime për cilat fjalë duhet të kemi më shumë kujdes dhe cilat mund të i pranojmë ashtu pa ju ditur kuptimin e plotë të tyre.

Po kjo dukuri paraqitet edhe gjatë mësimit të gjuhëve programuese, fjalë të reja 'bien në vesh' pandërprerë. Për disa nga këto fjalë duhet që ta urdhërojmë trurin, të tjerat regjistrohen vetvetiu nga truri ynë. Në mësimet pasuese do të mundohem që të kapë disa nga këto fjalë sipas përvojës që kam mbledhur nga librat, doracakët e ndryshëm që japin shembuj e mësime nga gjuha programuese C++.

Nga shkolla jemi të kalitur që shpjegimet për disa gjëra të komplikuara ti pranojmë ashtu si na jep mësuesi. Gjërat e komplikuara shpjegohen nga mësuesi duke ju përshtatur nivelit të njohurive të nxënësit (plot fjalë mbesin të pashpjeguara). Gjithë e dimë shpjegimin e mësuesit në klasën e parë: **1-2 = nuk bënë**, më vonë mësojmë se kjo bënë. Për këtë, mos lejo që terminologjitë e reja për ty, që përdoren në gjuhët programuese të turbullojnë njohurit nga shkolla. Shpesh ndodhë që edhe vetë shpikësit e terminologjive ende nuk e kanë të qartë funksionin e përkufizimit që kanë dhënë. Ndodhë që 'funksioni' të vjetrohet para se fjala përshkruese e tij, të ketë hyrë në literaturën e gjerë. Kjo nuk vlen vetëm për gjuhen shqipe por edhe më gjerë.

Pak histori

C++ (shqiptohet “cee plus plus”) është *statically typed, free-form, multi-paradigm, compiled*, gjuhë programuese me qëllim të përgjithshëm. Ajo konsiderohet si një gjuhë programimi e nivelit të mesëm (ang. intermediate-level), meqë ajo përbëhet nga një kombinim i tipareve të nivelit të ulët dhe të lartë. E zhvilluar nga Bjarne Stroustrup filluar në vitin 1997 në Bell Labs, ajo shton tipare të orientimit në objekte, siç janë klasat, dhe përmirësime të tjera të gjuhës programuese C. Fillimisht është quajtur **C me Klasa**, e cila u riemërua në C++ më 1983, duke përfshirë

edhe operatorin ritës. C++ është një nga gjuhët më të njohura programuese, dhe është zbatuar në një shumëllojshmëri të gjerë të harduerit dhe platformave të sistemit operativ.

Si një përpilues (ang. compiler) në kodin amtarë, fusha e tij e aplikimit përfshin softuer sistemet, sistemet aplikative, drejtuesit e pajisjeve (device drivers), serverët me performanca të larta (hight-performance) dhe klient aplikacionet, dhe softuerët argëtuese të tilla si video lojërat. Grupe të veçanta sigurojnë që të dy llojet free dhe regjistrues C++ përpilues softuerë, duke përfshirë Projektin GNU, Microsoft, Intel dhe Embarcadeero Technologies. C++ ka ndikuar në masë të madhe në shumë gjuhë programuese, më së shumti C# dhe Java. Gjuhë të tjera të suksesshme si C-Objektiv përdorin një sintaksë dhe qasje shumë të ndryshme për të shtuar klasa në C. C++ është përdorur gjithashtu për dizajnim të harduerit, ku dizajni fillimisht është përshkruar në C++, pastaj analizohet, në mënyrë arkitekturale i sforcuar, dhe planifikuar për të krijuar një gjuhë përshkruese regjistër-transfer (ang. regjistër-tranfer levelhardware description language) nëpërmjet sintezës së nivelit të lartë (ang. high-level)

Gjuha fillon si përmirësim i C, së pari duke shtuar klasa, pastaj funksione virtuale, operatorin e mbingarkuar, trashëgimi të shumëfishta, shabllone, përjashtimin e trajtimit ndërmjet tipareve tjera. Pas viteve të zhvillimit, gjuha programuese C++ u miratua si standarde në vitin 1998 si ***ISO/IEC 14882:1998***. Standardi është ndryshuar nga *2003 technical corrigendum*, ***ISO/IEC 14882:2003***. Standardi i tanishëm shtrinë C++ me tipare të reja të miratuara dhe botuara nga **ISO** në Shtator të vitit 2011 si ***ISO/IEC 14882:2011*** (e njohur zyrtarisht si C++ 11).

Për kë është ky libër

Ky libër u dedikohet studentëve të vitit të parë të fakultetit të informatikës apo shkencave kompjuterike si literaturë për lëndën *Programim I*. Gjithashtu këtë libër mund që ta përdorin edhe studentë të viteve tjera, nxënës të shkollave të mesme apo persona tjerë që dëshirojnë të mësojnë programimin në C++. Përfundimisht duke u bazuar në titullin, themi se me anë të këtij libri fillestarët mund të bëjnë një hap të sigurt drejtë botës së programimit, botë kjo e cila do t'ju sjell shumë suksese në të ardhmen.

Struktura e librit

Ky libër është ndarë në gjithsej 6 kapituj kryesorë: Code::Block& Visual Studio 2010, Baziket, Shprehjet, Degëzimet, Funksionet dhe Fushat, Pointerët dhe Referencat. Në fillim të librit kemi *parathënien* ku në të si fillim i është bërë një përshkrim programimit, pra gjegjësisht gjuhës programuese: Ç'është ajo? Si zhvillohet? Ç'far roli ka? Etj. Pastaj kemi *hyrjen* ku në të më së shumti jepet një përshkrim mbi nismën, apo ndjenjën fillestare në lidhje me programimin, si dhe mënyrën e të mësuarit të një gjuhe programuese kur ballafaqohemi për herë të parë. Në vazhdim kemi *pak histori* mbi fillimin e gjuhës programuese C++, zhvilluesin e sajë, avantazhet e kësaj gjuhe programuese, si dhe në fund kemi standardet për këtë gjuhë. Kështu më pas pasojnë *paradigmat e programimit*, ku me anët të këtij përshkrimi i ofrohet lexuesit një njohuri më detale në lidhje me programimin dhe stilet e tij. Pas një përshkrimi të shkurtër *mbi autorin* në fund kemi edhe një pjesë tjetër pra atë *për kë është ky libër*, pjesë kjo e cila na përshkruan me disa fjalë se për cilën audiencë është ky libër. Shkurt kush mund ta përdor këtë libër. Në vazhdim kemi një përshkrim të shkurtër për çdo kapitull.

Kapitulli I: Në këtë kapitull kemi hyrjen në programim, ku më saktësisht flitet për atë se çka është programimi, kategoritë e gjuhëve të programimit, rregullat e paraqitjes së algoritmeve dhe paradigmat e programimit.

Kapitulli II: Në këtë kapitull kemi një përshkrim detal në lidhje me mënyrën se si ti ekzekutojmë këto programe në kompjuter. Këtu janë zgjedhur dy mënyra: *e para* duke shfrytëzuar programin **CodeBlock** dhe *e dyta* duke shfrytëzuar paketën e kompanisë Microsoft **Visual Studio 2010 (C++)**.

Kapitulli III: në këtë kapitull kemi elementet bazike të gjuhës programuese C++. Po ashtu kemi edhe programin e parë fillestarë, pra kemi të sqaruar mënyrën e shkrimit të programeve në këtë gjuhë.

Kapitulli IV: në këtë kapitull kemi të bëjmë me shprehjet e ndryshme në këtë gjuhë programuese, siç janë operatorët (aritmetik, logjik, relativ, i shoqërimit, operatoret e pavlefshëm, ritës/zvogëlues, e caktimit etj.) vlerat logjike, konvertimin e tipave etj. Në përgjithësi mund të themi se në këtë kapitull kemi të bëjmë me pjesën matematikore që na ofron kjo gjuhë programuese.

Kapitulli V: na paraqitet pak si më i avancuar, në të cilin kemi degëzimet, ciklet të mundshme në këtë gjuhë programuese, ku janë

përshkruar thuajse të gjithë komandat për degëzim: if, for, while, do, switch, continue, brake, goto dhe return.

Kapitulli VI: këtu kemi të bëjmë me funksionet dhe llojet e ndryshme të paraqitjes së tyre, shtrirjen në program, operatorin e fushës, ndryshoret (automatike, regjistruese, eksterne, statike etj.), rekursivin, numëruesin, runtime stack etj.

Kapitulli VII: ky kapitull na paraqet fushat (si më të rëndësishme), pointerët dhe referencat. Tek fushat kemi vektorët dhe matricat, memorien dinamike, pointerët aritmetik, komandën typedef etj.

Në fund kemi pjesën falënderuese ndaj përdoruesve të këtij libri dhe referencat ku paraqiten burimet që janë shfrytëzuar gjatë përpunimit të librit.

Gjithashtu është me rëndësi të ceket se në çdo nënkapitull kemi shembuj që janë ilustruar dhe në fund të çdo kapituj kemi nënkapitullin *shembuj* dhe *kontrollo njohuritë.* Në të parin kemi shembuj të zgjidhur në *Code::Blocks* ndërsa në të dytin kemi shembuj të pa zgjidhur që shërbejnë si njohuri për studentin.

Kapitulli I

Hyrje në programim

Nënkapitujt:

- ✓ **Ç'far është programimi**
- ✓ **Kategoritë e gjuhëve të programimit**
- ✓ **Paradigmat e programimit**
- ✓ **Cikli jetësor i zhvillimit të programit**
- ✓ **Rregulla të paraqitjes së algoritmeve me flowchart**
- ✓ **Shembuj**
- ✓ **Kontrollim njohurishë**

Ç'fare është programimi?

Nocioni programim është kompleks dhe përfshin një sere shpjegimesh dhe atë:

- Programimi është procesi i shkrimit të sekuencave të instruksioneve të cilat do të ekzekutohen nga kompjuteri për të zgjidhur një problem të dhëne duke e ndarë në hapa më të vegjël të thjeshtë. Zgjidhja e këtyre hapave na ndihmon për zgjidhje finale të problemit.
- Programuesit, janë njerëzit që shkruajnë programe kompjuterike duke përdorur gjuhë të programimit për ti komunikuar instruksione kompjuterit.

Duke qenë se **kompjuterët veprojnë në një mënyrë jo logjike** (ndryshe nga mendja e njeriut) atyre u duhet ta ndajnë një detyrë në pjesë më të vogla. Për ta kuptuar marrim një shembull: si mund t'i vendosim këto tre fjalë në rend alfabetik?

Amerika, Zelanda, Anglia?

1. Shohim fjalët që fillojnë me "A"
2. Nëse gjejmë ndonjë fjalë me "A" e vendosim në fillim të listës
3. Shohim nëse ka ndonjë fjalë tjetër që fillon me "A"
4. Nëse gjejmë ndonjë fjalë tjetër që fillon me "A", krahasojmë shkronjën e dytë të fjalëve
5. Nëse shkronja e dytë e fjalës që gjetëm është e ndryshme me fjalën e parë, e vendosim në rend alfabetik
6. E përsërisim të njëjtën gjë me fjalët që fillojnë me "B", derisa të gjitha fjalët të vendosen në rendin e tyre.

Ajo që vëmë re është se sa kohe na u desh për të bërë këtë veprim. Juve nuk ju desh të uleshit e te hartonit një plan se si do ta bënit këtë gjë, duke shpjeguar në detaje hapat e mësipërm.

Një shembull tjetër. Numëroni sa fjalë ka në këtë fjali: *Lënda e programimit është argëtuese*

Së pari na duhet të shpjegojmë se ç'fare është fjala. Fjala është një vazhdimësi shkronjash e cila ndahet me anë të hapësirës (space) nga

shkronja tjetër. Duke iu përmbajtur këtij rregulli fjalia e mësipërme ka 5 fjalë.
Po në fjalinë e mëposhtme, sa fjalë ka?

"Disa njerëz pëlqejnë të shkruajnë fjali të, shkurtra"

Në këtë fjali ka 8 fjalë por kjo nuk i bindet rregullit tonë, që do të thotë në fjali ka 7 fjalë. Nëse ne e rishikojmë dhe njëherë rregullin tone, atëherë do të themi që një fjalë ndahet dallohet nga tjetra me anë të hapësirës, presjes ose vizës së mesit.

Kompjuterët nuk janë të mençur

Kompjuterët bëjnë atë që u jepet për të bërë, sado e gabuar qoftë. Ashtu sikurse mund të fshijnë pa problem edhe hard diskun e vet. Ata kërkojnë të udhëhiqen në çdo detaj të vogël, madje se si të bëjnë edhe veprimin më të thjeshtë. E vetmja gjë pozitive është se kompjuterët janë totalisht të bindshëm. Nuk ka rëndësi se sa pa kuptim mund të jenë instruksionet e tua, kompjuteri do t'i kryejë ato në mënyrë të saktë.

Tani shohim se si këto lidhen me njëra-tjetrën. Programuesit me ane te instruksioneve i tregojnë kompjuterëve se ç'fare të bëjnë. Kompjuterët kërkojnë instruksione precize dhe të plota. Truri i njeriut ndodh që jo gjithmonë ti japë instruksione të sakta vetvetes por edhe nëse i jep komanda të mangëta, merr përsëri përgjigje korrekte, e njëjta gjë nuk ndodh me kompjuterin. Kështu që programuesit duhet të mendojnë në mënyrë jo logjike. Ata duhet të mësojnë se si të japin përshkrimin e një detyre (p.sh. numërimi i fjalëve në një fjali), dhe të rendisin hapat kryesor, të cilët i duhen kompjuterit me qëllim që të përmbush atë detyrë.

Kategoritë e gjuhëve të programimit

Programorët shkruajnë instruksionet në gjuhë të ndryshme programimi, disa direkt të kuptueshme nga kompjuterët dhe të tjerat me hapa përkthimi ndërmjetësues. Ekzistojnë qindra gjuhë programimi në përdorim në ditët e sotme të ndara në tre kategori kryesore:

Gjuhë programuese e makinës (*Machine languages*)

Çdo kompjuter kupton direkt vetëm gjuhën e makinës të vetë, kjo gjuhë është *gjuha natyrale* e një kompjuteri specifik e përcaktuar nga ndërtimi fizik (*hardware*) i tij, me pak fjalë këto gjuhë janë specifike në varësi të vetë njësisë qendrore (CPU). Përbëhet kryesisht nga vargje numrash të shprehura në *sistemin binar* të cilat udhëzojnë kompjuterët të kryejnë në mënyrë të njëpasnjëshme operacionet e tyre themelore. Këto gjuhë programimi janë të rënda për programorët në sensin e kohës së tepërt që kërkojnë për të programuar qoftë edhe gjënë më të thjeshtë. Një program i shkruar në këtë gjuhë do të kishte një pamje të këtillë:

```
10110011 00011001
01111010 11010001 10010100
10011111 00011001
01011100 11010001 10010000
10111011 11010001 10010110
```

Gjuhë programimi të nivelit të ulet (*Low-level/Assembly languages*)

Këto gjuhë programimi janë të njëjta me gjuhët që vetë makina kupton, por me një ndryshim të vogël në përdorimin e emrave në vend të vargjeve numerike për operacione elementare. Çdo instruksion transformohet në një instruksion makine nga një program përkthimi specifik i quajtur *Assembler*.

Një program i shkruar në gjuhën asembler ka një pamje të tillë:

```
MOV 0, SUM
MOV NUM, AC
ADD SUM, AC
STO SUM, TOT
```

Gjuhë programimi të nivelit të lartë *(High-level languages*)

Një gjuhë programimi e nivelit të lartë është ajo e cila është familjare (e kuptueshme nga qenia humane), e pavarur nga ambienti i aplikimit dhe në një far mënyre, abstrakte në atë ç'farë ndodh në realitet në njësinë qendrore (CPU). Duke qenë se këto gjuhë programimi janë

shumë familjare me programorin, janë shumë të kuptueshme nga ai, bën që një rregull i shkruar në këto gjuhë të transformohet në një numër të madh instruksionesh të interpretuara nga vetë makina (në rastin tonë nga njësia qendrore) nëpërmjet një programi përkthimi specifik i quajtur *Compiler*. Një program i shkruar në gjuhë të nivelit të lartë mund të duket si më poshtë:

```
#include <iostream>
int main()
{
        cout << "Hello World!";
}
```

Në këtë kategori bëjnë pjesë gjuhë programi si: **C, C++, Java, Basic, Fortran** etj. Le të flasim shkurtimisht për secilën gjuhë programimi.

C

Kjo është gjuha më e përdorur e programimit, dhe më e vjetra nga gjuhët tjera. Është përdorur që në vitet 70, dhe është një gjuhë që kuptohet mirë (do të thotë që ka një literaturë të gjerë dhe shembuj të kodeve të programimit). Kjo rekomandohet si një gjuhë bazë programimi.

C++

Kjo gjuhë e përfshin C (pra C e evoluar ose e shtuar me opsione të reja). Avantazhi i kësaj gjuhe programimi është orientimi në objekte ndryshe nga C që është një gjuhë procedurale. Gjuhët e orientuara në objekte janë më moderne dhe më të përdorura në botën e programimit. C është në qendër të C++, që do të thotë se duhet të dini C për të përdorur C++. Është gjuha e dytë më e përdorur pas C, që së shpejti do të bëhet gjuha më e përdorur.

Java

Java ka një avantazh që gjuhët e tjera të programimit nuk e kanë. Një program i krijuar në Java ekzekutohet në sisteme shfrytëzimi (platforma) të ndryshme pa qenë e nevojshme të rikompilohet. Platforma është një lloj i veçantë i sistemit të shfrytëzimit siç janë Windows, Mac OS ose Linux. Normalisht, nëse dëshironi të përdorni të njëjtin program në një platformë të ndryshme nga ajo ku është shkruar, do ju duhet ta

rikompiloni. Ndonjëherë ju duhet të ndryshoni disa kode për t'iu përshtatur platformës së re. Java mund të ekzekutojë në më shumë se një platformë pa nevojën e rikompilimit. Ja pse ky është një avantazh. Por në të njëjtën kohë Java ka dhe një mangësi që mund të konsiderohet serioz: është i ngadaltë. Java arrin ta bëjë këtë marifet duke vendosur një program në kompjuter, që konsiderohet si një kompjuter në vete. Java vepron brenda kësaj makine virtuale (virtual machine) e cila ekzekutohet në çdo platforme (p.sh. Windows ose Mac OS).

Paradigmat e programimit

Paradigma programore paraqet stilin thelbësor të programimit kompjuterik. Paradigma dallon nga metodologjia nga se metodologjia paraqet stilin për zgjidhjen e një problemi specifik në inxhinieringun softuerik. Paradigmat dallojnë nga njëra tjetra për nga konceptet dhe abstraksionet që i shfrytëzojnë për paraqitjen e elementeve të programit (si p.sh. objektet, funksionet, variablat, etj.) si dhe dallojnë për nga hapat që i marrin për përpilimin e një llogaritjeje (p.sh. vlerësimi, rrjedhshmëria e të dhënave, etj.)

Një gjuhë programuese mund të përkrah paradigma të shumëfishtë. Për shembull, programet që shkruhen në C++ ose Object Pascal munden të jenë vetëm procedurale, ose vetëm të orientuara në objekte, mirëpo këto programe munden të përmbajnë edhe elemente tët dy paradigmave. Se cilat elemente të paradigmave përfshihen në program varet tërësisht nga dizajnerët e softuerit.

Në **Programimin e Orientuar në Objekte**, programorët mund ta perceptojnë paradigmën si një koleksion të objekteve ndërvepruese; përderisa në **Programimin Funksional**, programi mundet të perceptohet si një sekuencë të funksioneve vlerësuese.

Disa gjuhë programuese përkrahin një paradigmë të vetme, si **Smalltalk** që përkrah vetëm *Programimin e Orientuar në Objekte*, ose **Haskell** që përkrah vetëm *Programimin Funksional*. Mirëpo, ekziston një numër i gjuhëve programuese që përkrahin paradigma të shumta, si p.sh. **Object Pascal, C++, C#, Visual Basic, Common Lisp, Scheme, Python, Ruby** dhe **Oz**.

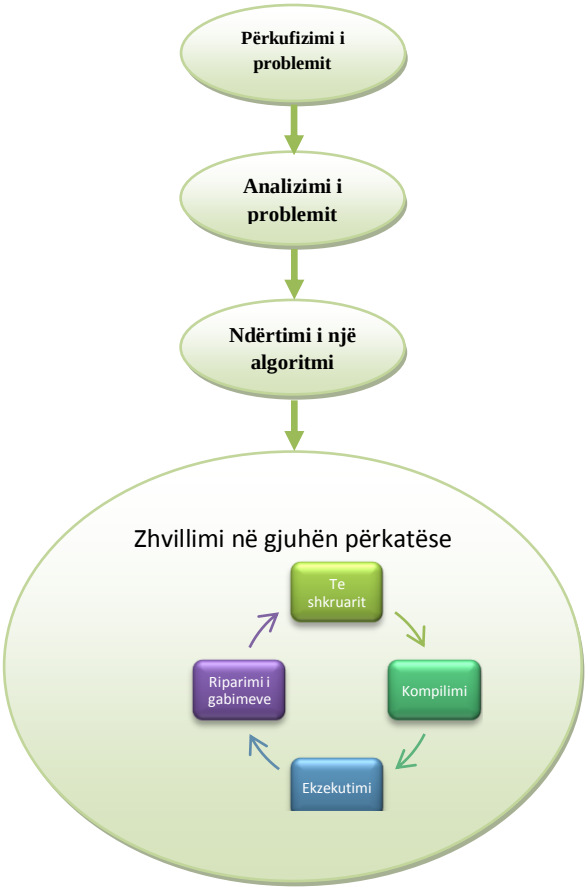
Paradigmat e programimit janë të njohura gjithashtu edhe për nga teknikat që nuk i lejojnë, si p.sh. në *Programimin Funksional* nuk është i lejuar shfrytëzimi i efekteve anësorë (ku kjo do të thotë që funksioni përveç që e bënë kthimin e një vlere, ai gjithashtu bën ndryshimin e ndonjë gjendjeje, p.sh. ndryshohet ndonjë variabël globale ose statike, ose shkruhen të dhëna në një skedë, etj.). Në anën tjetër, *Programimi i Strukturuar* nuk e lejon shfrytëzimin e deklarimit **goto**.

Cikli jetësor i zhvillimit të programit

Ka disa faza në të cilat kalon zhvillimi i një programi, duke ndjekur një plan ose metodologji të organizuar mirë e cila ndan procesin e punës në një seri punësh më të vogla.

Hapat kryesorë në zgjidhjen e një problemi nga kompjuteri janë:

1. Përkufizimi i problemit
2. Analizimi i problemit
3. Ndërtimi i një algoritmi dhe
4. Zhvillimi i në gjuhën përkatëse.



Përkufizimi i problemit

Para se të zhvillohet programi për një problem të dhënë duhet që problemi të jetë përcaktuar qartë dhe mirë për sa i përket asaj që merr në hyrje (*input*) dhe asaj që jep në dalje (*output*). Një mirë i përcaktuar është gjysma e zgjidhjes së tij.

Konsiderojmë problemin : Krijoni një program i cili afishon numrin total të prezencës të një emri në një fjali të dhënë.

Analizimi i problemit

Pasi problemi është përcaktuar në mënyrë të qartë, duhet formuluar zgjidhja në mënyrën më efektive të mundshme. Ky hap përfshin ndarjen e problemit kryesorë në nën probleme të thjeshta, më të vogla.

Problemi: Afisho numrin total të prezencës të një emri në fjali.

Input: Fjalia e dhënë, emri që kërkohet.

Output: Numri total i prezencës të emrit në fjalinë e dhënë.

Ndërtimi i një algoritmi

Zgjidhja e një problemi vjen pasi vetë problemi është përcaktuar dhe analizuar në mënyrë tepër të qartë, për këtë në programim kërkohet që zgjidhja e problemit të shprehet hap pas hapi.

Algoritmi është një procedurë e përbëre nga një bashkësi e fundme rregullash të qarta dhe mirë të specifikuara e cila përcakton një sekuencë operacionesh të fundme që çojnë në zgjidhjen e një problemi. Në këtë pikë mund të themi që programi është të shprehurit e një algoritmi në një gjuhë kompjuteri. Këto hapa mund të shprehen me fjalë ose në mënyrë grafike nëpërmjet skemave grafike (*flowchart*) ose në mënyrë tekstuale ose analitike me *pseudocode* e cila është një ndërthurje në mes gjuhës së folur humane dhe asaj formale të programimit. Kjo mënyrë shprehjeje na ndihmon për të kuptuar algoritmin para se ta shkruajmë atë në ndonjë gjuhë programimi.

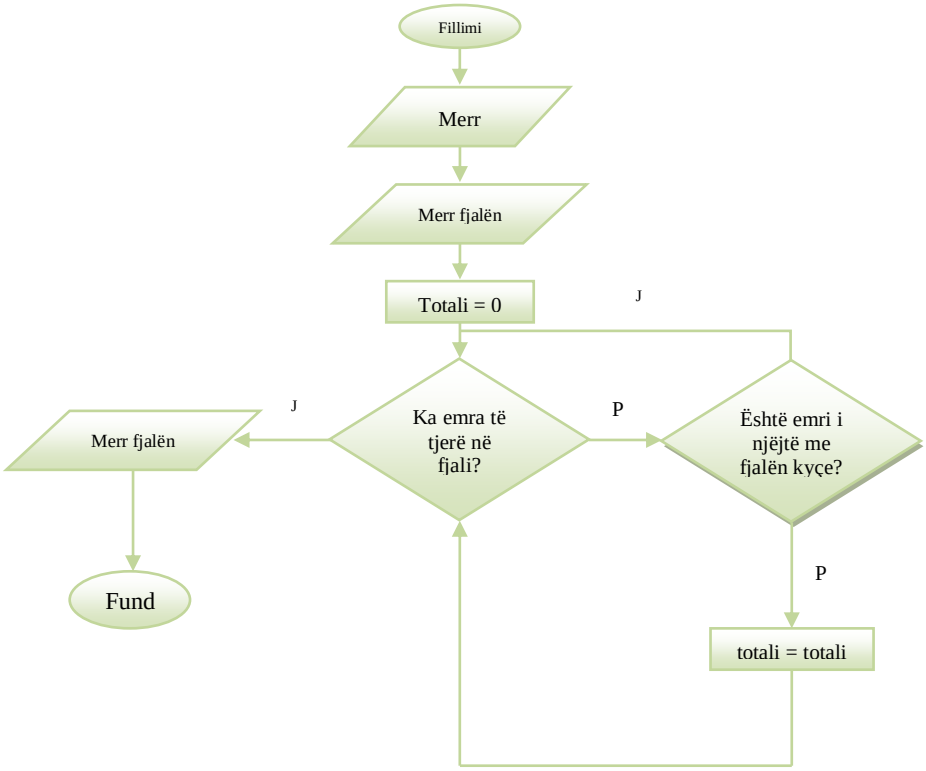
Ndërkohe konsiderojmë paraqitjen e problemit të më sipërm përmendur në një

mënyrë të tillë që të jetë ende i kuptueshëm, pa humbur përmbajtjen.

Paraqitja e algoritmit me fjalë:

1. Konsidero fjalinë e dhënë.
2. Konsidero emrin e kërkuar, le ta quajmë atë si fjalë kyçe.
3. Krahaso fjalën kyçe me të gjitha emrat në fjali.
4. Në qoftë se fjala kyçe është e njëjtë me ndonjë emër në fjali, shtoji 1 totalit.
5. Në qoftë se të gjithë emrat në fjali janë krahasuar, afisho rezultatin, në rastin tonë totalin.

Paraqitja e algoritmit me flowchart:



Paraqitja e algoritmit me pseudocode:

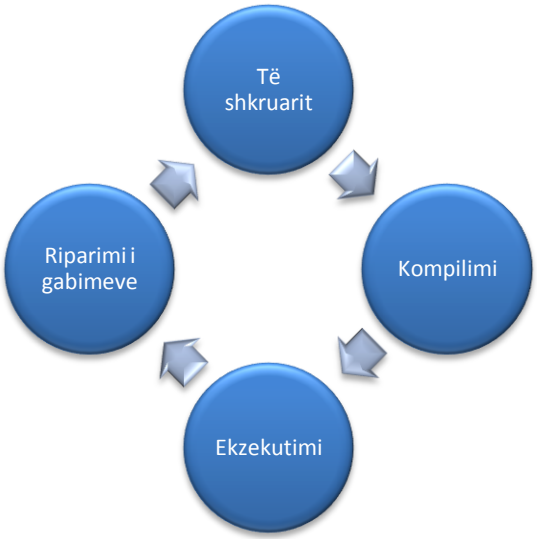
1. listaEmrave = fjalia me emrat
2. fjalaKyce = emri që kërkohet
3. totali = 0
4. Për çdo emër në listaEmrave vazhdo më poshtë
5. Në qoftë se emer == fjalaKyce
6. totali = totali + 1
7. Afisho totali

Zhvillimi në gjuhen përkatëse

Pas ndërtimit të algoritmit, është e mundur krijimi i kodit programues (*source code*). Duke përdorur algoritmin si bazë, kodi programues mund të shkruhet në njërën nga gjuhët e programimit më të përshtatshme. Procesi i programimit kalon në fazat e mëposhtme:

1. Të shkruarit e programit (*Program writing)*,
2. Kompilimi i programit (Program compiling),

3. Ekzekutimi i programit (*Program running*),
4. Riparimi i gabimeve të programit (*Program debugging*),
5. Përsëritja e gjithë procesit derisa programi të konsiderohet i përfunduar.



Le ti diskutojmë këto hapa një nga një:

Të shkruarit e programit

Ju nuk mund të shkruani një program në gjuhën e folur. Pra ju nuk mund të shkruani në kompjuter "numëro numrin e fjalëve në fjali". Për këtë nevojitet një **gjuhë programimi.** Në gjuhën e programimit përdoren fjalët si "if", "repeat", "end", dhe operatorët matematikorë "+" apo "=". Pra është një çështje e të mësuarit të "gramatikës" së kësaj gjuhe; mënyra e të thënies saktë të gjërave.

Të shkruash një program do të thotë të shkruash hapat që nevojiten për të kryer një punë (detyrë), duke përdorur gjuhën e programimit që dini. Të shkruarit e programit do të bëhet në një ambient programimi (*Programming enviroment*). Ka ambiente programimi të ndryshme për të shkruar këto programe në varësi të gjuhës së programimit që zgjidhet.

Siç është përmendur më lartë, ajo që shkruhet për të krijuar një program quhet *kod programimi* (*source code*) ose thjeshtë *kod* (*code*). Programuesit shpesh e quajnë edhe *"programming coding"*.

Kompilimi i programit

Që një program i shkruar në gjuhë të nivelit të lartë të punojë në një kompjuter duhet që të përkthehet në gjuhën e makinës. Ka dy lloje përkthyesish (*Translator*): kompiluesit dhe interpretuesit, dhe gjuhët e nivelit të lartë quhen ose gjuhë të kompiluara ose gjuhë të interpretuara.

Kompiluesi (*Compilator*) është një program përkthimi i cili përkthen programin e shkruar në gjuhë të lartë të makinës, i njohur si *source code*, në gjuhën e makinës, i njohur si *object code*. Procesi i përkthimit quhet kompilim.

Interpretuesi (*Interpreter*) është një program i cili përkthen instruksionet e programit, resht pas reshti në gjuhën e makinës, pak para se instruksioni i programit të ekzekutohet. Përkthimi dhe ekzekutimi ndodhin në mënyrë të menjëhershme, njëri pas tjetrit, një deklaratë (*statemant*) në një kohë. Ndryshe nga gjuhët e kompiluara, asnjë *object code* nuk ruhet dhe nuk ka kompilim lidhur me të. Kjo do të thotë se në qoftë se në një program kemi një deklaratë që duhet të ekzekutohet disa herë me radhë atëherë kjo deklaratë duhet të përkthehet në gjuhën e makinës çdoherë kur ekzekutohet. Gjuhët e kompiluara janë më të mira se sa gjuhët e interpretuara, meqenëse ato mund të ekzekutohen më shpejtë dhe në mënyrë më efikase. Nga ana tjetër, gjuhët e interpretuara nuk kanë të nevojshme të krijojnë *object code* kështu që përgjithësisht janë më të thjeshta për tu zhvilluar, gjegjësisht, koduar dhe testuar.

Ekzekutimi i programit

Mbas kompilimit të programit, duhet të shohim nëse funksionon, pra duhet ta bëjmë kompjuterin të kryejë hapat që specifikuam. Kjo quhet ekzekutim i programit, njësoj si një makinë që nuk punon nëse nuk e nget, ashtu edhe programi nuk punon nëse nuk e ekzekuton.

Riparimi i gabimeve të programit

Termi *debug* i referohet riparimit të gabimeve dhe problemeve të programit. Zakonisht procesi i *debug* bëhet duke ndjekur ecurinë e ekzekutimit të programit hap pas hapi në kodin e tij.

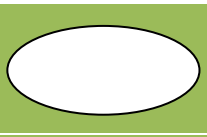
Përsëritja e gjithë procesit derisa programi të ketë përfunduar

Më pas behet përsëritja e të gjithë procesit derisa programi të ketë përfunduar. Ndodh që mund t'i rikthehesh programit, të riparosh gabimet, të bësh përmirësime, dhe të ndryshosh kodin ekzistues.

Pasi kemi parë të gjitha fazat në të cilat kalon zgjidhja e një problemi të dhënë, do rikthehemi të studiojmë në detaje rregullat e paraqitjes grafike të algoritmit nëpërmjet flowchart-eve. Në vazhdim do të paraqesim, jo të detajuara, në mënyrë grafike strukturat kryesore të kontrollit të një programi.

Rregullat e paraqitjes së algoritmeve me flowchart

Ekzistojnë pesë simbole kryesore që përdoren në flowchart-e. Lista e më poshtme nuk paraqet të gjithë simbolet që përfshijnë flowchart-et, por vetëm ato më kryesore për të përshkruar algoritmet e problemave të thjeshta.

Simboli	Emri	Funksioni
	Terminal	Tregon fillimin/ mbarimin e programit ose procesit.
	Process	Përdoret për deklarim të dhënash, veprime të ndryshme matematikore, logjike etj.
	Input/Output	Përdoret për çdo tip operacioni Input/Output.
	Decision	Përdoret për veprime logjike, krahasimi etj.
	Flow Lines	Tregon drejtimin e rrjedhjes së operacioneve.

Rregulla të përgjithshme:

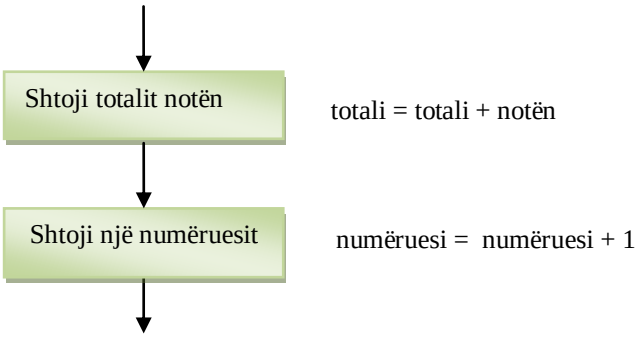
1. Të gjithë simbolet janë të lidhura me shigjeta duke përdorur simbolin *Flow lines*.
2. Çdo simbol ka vetëm një pikë hyrëse, përveç rastit kur përdoret simboli *Terminal* për të treguar fillimin gjithashtu dhe një pikë dalëse përveç rastit të simbolit *Terminal* për të treguar mbarimin e programit si dhe simbolit *Decision.*
3. Simboli *Decision* ka dy pika dalëse në varësi të rezultatit.
4. **Flowchart-i** fillon me simbolin *Terminal* "Start" dhe përfundon me atë "End" për të nënkuptuar fillimin dhe mbarimin e programit.

Strukturat e kontrollit

Normalisht deklaratat në një program ekzekutohen njëra pas tjetrës në radhën që janë shkruar, ky quhet ekzekutim sekuencial (*sequencial execution*). Ka dhe deklarata në të cilat hapi i radhës mund te mos jetë ai i mëposhtmi por një tjetër, ky quhet transferim i kontrollit (*control transfer*). Të gjithë programet ose algoritmet mund të shkruhen vetëm me anë të tre strukturave të kontrollit të cilat janë:

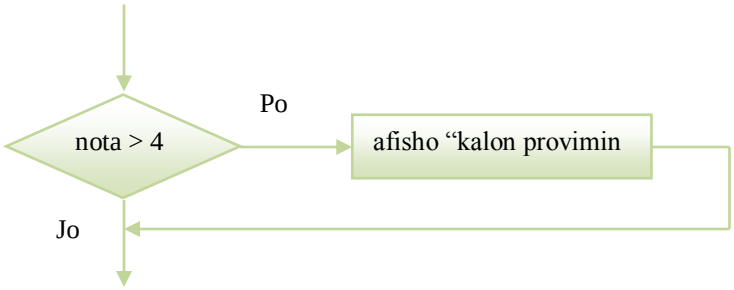
Struktura te renditura (Sequence structure)

Këto struktura nëse nuk kanë deklarata për transferimin e kontrollit, në mënyre thelbësore ekzekutimi i tyre vazhdon në radhë një pas një siç janë shkruar. Kjo strukturë mund të paraqitet në flowchart si më poshtë:



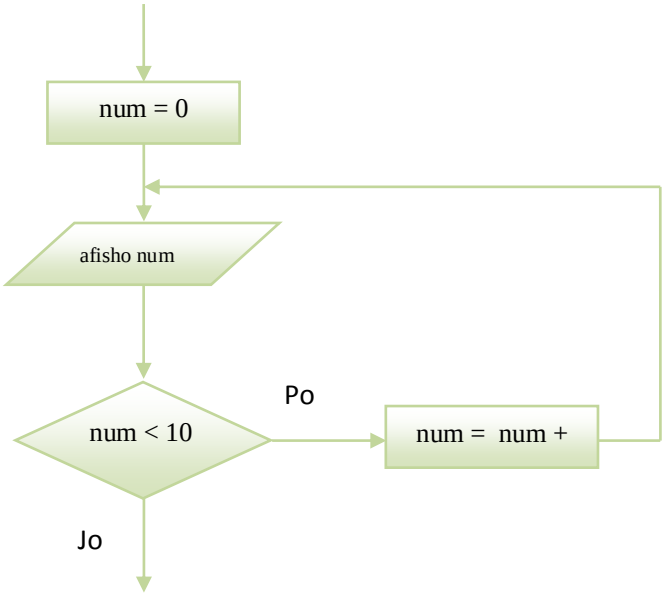
Struktura përzgjedhëse (Selection structure)

Këto struktura përdoren për të zgjedhur mënyra alternative të veprimit në bazë të një kushti të dhënë. Konsiderojmë strukturën e më poshtme e cila afishon "kalon provimin" nëse nota është më e madhe se 4.



Struktura përsëritëse (Repetition structure)

Kjo strukturë të lejon të specifikosh që një operacion duhet ripërsëritur derisa një kusht mbetet i vërtetë. Konsiderojmë strukturën e më poshtme e cila afishon numrat nga 0 deri në 9.



Kapitulli 2

Code Blocks & Visual Studio

Nënkapitujt:

- ✓ **Code Block**
  - **Instalimi i Code Blocks**
  - **Përdorimi i Code Blocks**
- ✓ **Vizual Studio**
  - **Instalimi i Vizual Studios**
  - **Përdorimi i Vizual Studios**

Code Blocks

Code::Blocks është burim i hapur dhe i lirë (free), ndër-platformë IDE që përkrahë kompajler të shumtë përfshirë GCC dhe MSVC. Është e zhvilluar në C++ duke përdorur wxWidgets si vegël e GUI. Duke përdorur një arkitekturë plugin kapacitetet dhe karakteristikat e saja janë të përcaktuara nga plugin-ët e ofruar. Aktualisht, Code::Blocks është e orientuar drejt C dhe C++. Gjithashtu mund të përdoret edhe për krijimin e ARM, AVR, D, DirectX, FLTK, Fortran, GLFW, GLUT, GTK+, Irrlicht, Lightfeather, MATLAB, OGRE, OpenGL, Qt, SDL, SFML, STL, SmartWin dhe wx programeve dhe aplikacioneve, edhe pse në disa raste është i nevojshëm instalimi i SDKs ose framework-it.

Code::Blocks është duke u zhvilluar për Windows, Linux and Mac OS X dhe ka qenë i mbajtur në FreeBSD.

Versioni i fundit stabil si Gusht 2012, Code::Blocks 10.05, është lirua më 30 Maj 2010.

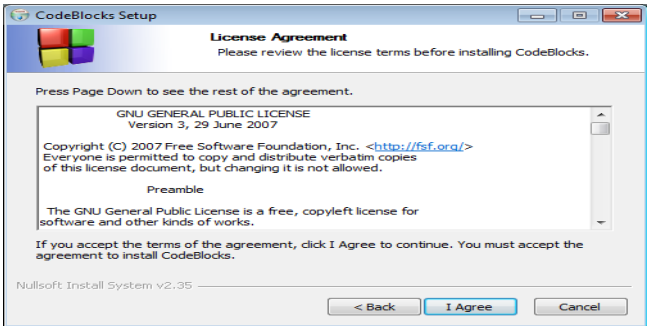
Instalimi

Në këtë kapitull do të mësoni se si të instaloni dhe përdorni programin CodeBlock për të shkruar programet në C++. Në fillim e shkarkojmë këtë program nga ky link: http://www.codeblocks.org/downloads. Ky program është burim i lirë.

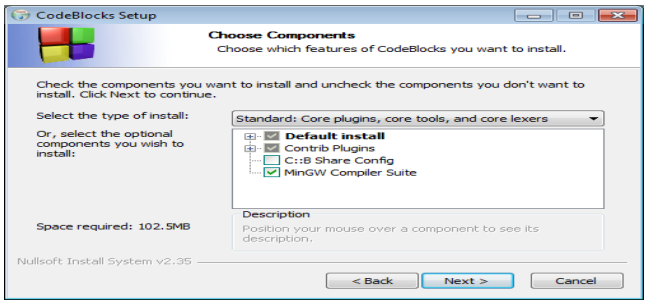
Pasi të shkarkoni e hapni *setup* – in përkatës.

Pasi të bëhet *extract*, në ekran do të paraqitet dritarja në vazhdim:

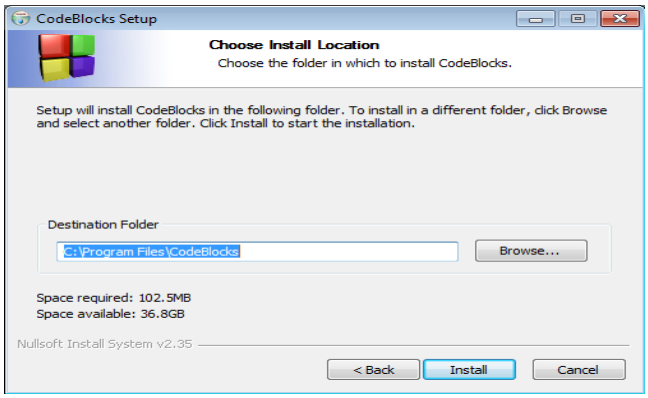
Klikojmë **Next >** dhe më pas na paraqitet dritarja e më poshtme:



Tani shtypim sustën **I Agree** dhe kalojmë tek dritarja në vazhdim:



Në këtë dritare nuk prekim gjë, lëmë që të instalohet tipi Standard dhe klikojmë **Next >**. Paraqitet dritarja në vazhdim:

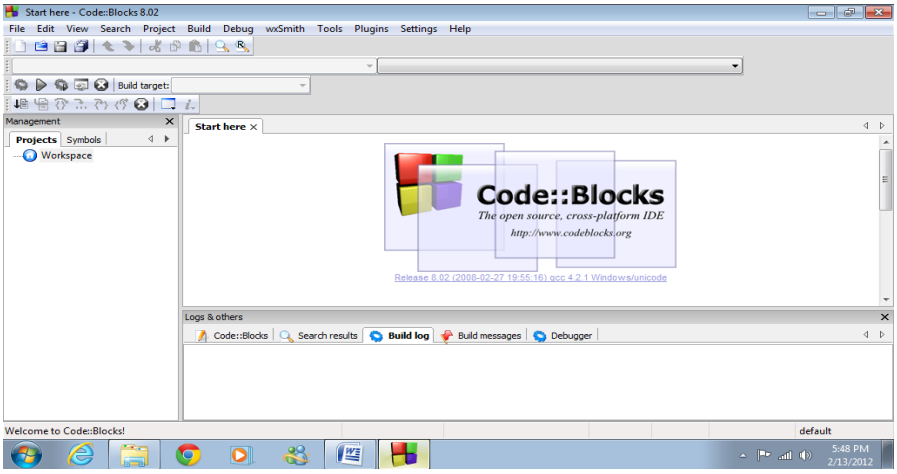


Tani më jemi në dritaren e fundit të instalimit, ku tek **Browse..** mund të zgjedhim lokacionin se ku dëshirojmë të ruajmë programin, pastaj shtypim **Install**. Pas kësaj do të fillon instalimi i programit dhe pasi të ketë përfunduar shtypim sustën **Close**.

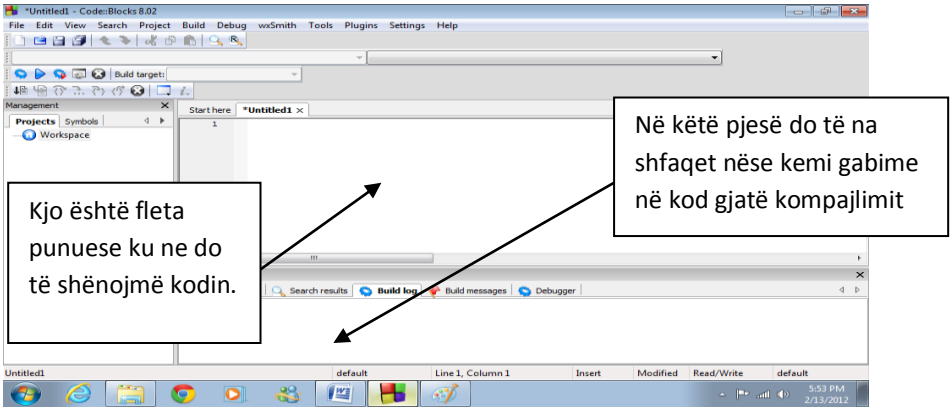
Startimi i Code Block

Pas instalimit të programit, në desktop është krijuar ikona e CodeBlock dhe klikojmë aty dy herë që të starton programi.

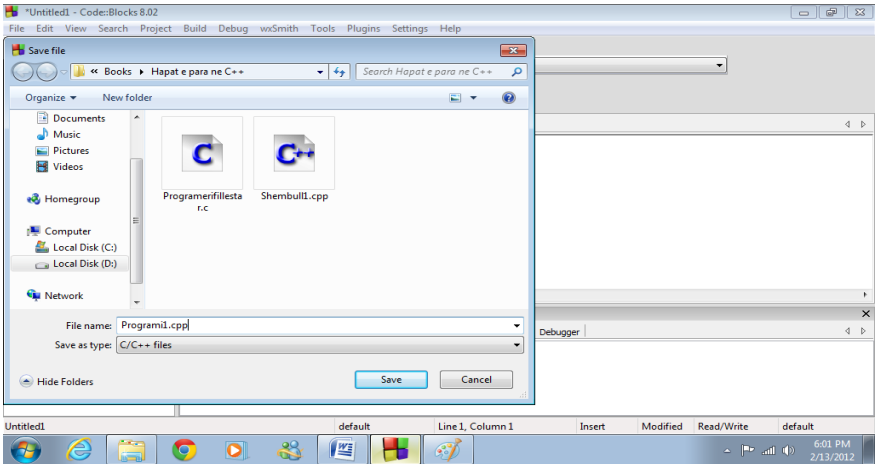
Startimi i programit i cili në fillim duket si në figurën e më poshtme.



Tani për të krijuar një krijuar një fajll c++, shkojmë tek menyja rënëse **File > New > Empty File**, pastaj në program do të kemi këtë pamje:

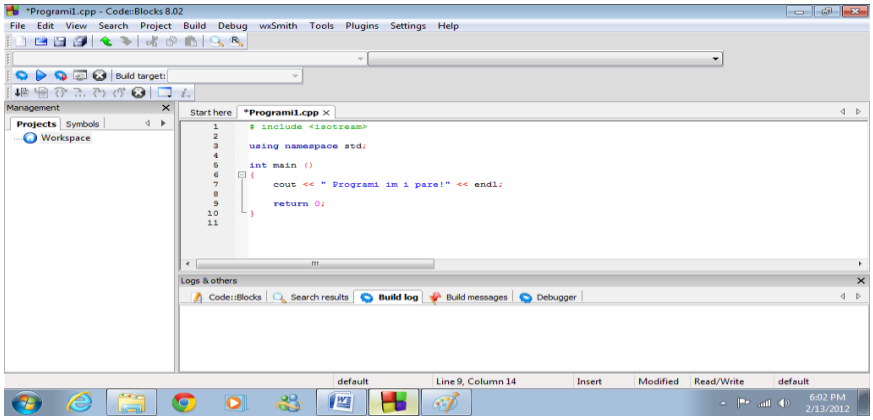


Pasi të kemi mbërritur deri te ky hap, tani do të duhet të bëjmë ruajtjen e këtij fajlli në disk. Shkojmë në menynë rënëse **File > Save as...** Tani na paraqitet dritarja për ruajtje të fajllit që duket si më poshtë:



Siç shohim edhe në figurë ne mund të ruajmë këtë fajll ku të duam, por kujdes duhet tek prapashtesa, e cila duhet të jetë cpp (shih foton).

Pasi të kemi regjistruar fajllin, ne mundemi të shkruajmë kodin përkatës:

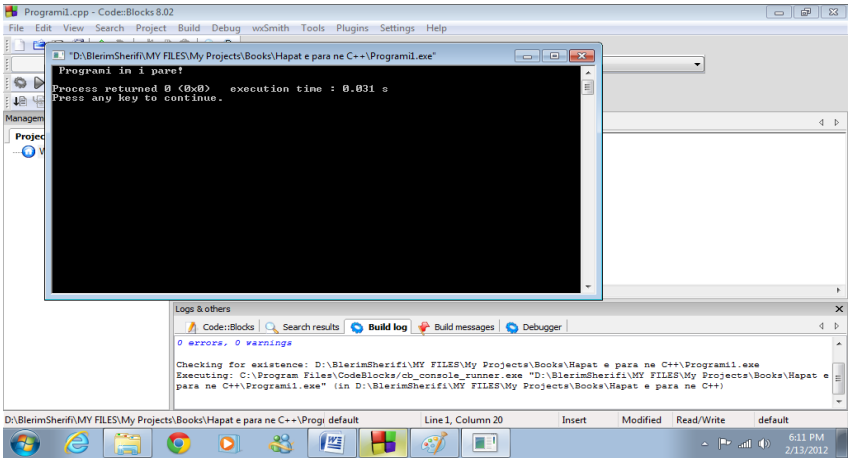


Siç shihet në figurë fjalët kyçe të gjuhës C++ janë të ngjyrosura me ngjyrë, që na lejon neve të kuptojmë se ato janë të rezervuara për gjuhën C++. Pasi të kemi përfunduar me shkrimin e kodit, e kompilojmë atë ose direkt e kompilojmë dhe e ekzekutojmë.

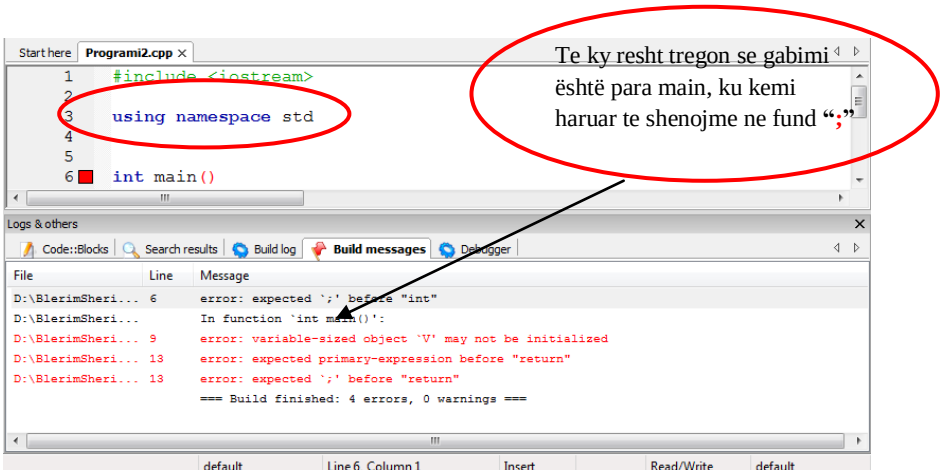
Për ta kompiluar shkojmë në menynë rënëse **Build > Build** ose shtypim kombinimin e tasteve **Ctrl + F9**.

Për ta kompiluar dhe Ekzekutuar njëkohësisht, shkojmë në menynë rënëse **Build > Build and run** ose shtypim tastin **F9**.

Pastaj në ekran do të na paraqitet teksti “Programi im i pare!”, si në foton në vazhdim:



Në rast të gabimeve kompajleri na njofton për gabimet në fjalë dhe ato mund ti shohim në dritaren **Logs & others** që shihet në figurën e më poshtme:



Visual Studio

Microsoft Visual Studio është një softuer i integruar për zhvillim programesh, web aplikacione dhe aplikacione tjera të avancuara. Ky softuer është një produkt nga kompania Microsoft.

Visual Studio mbështet gjuhë programuese të ndryshme me anë të shërbimit të gjuhës, e cila lejon editorin e kodit dhe debagerin (program për gjetjen e gabimeve ne kodin e shkruar) të mbështet pothuajse ndonjë gjuhë programuese, duke siguruar se shërbimet e gjuhës specifike ekzistojnë. Këto gjuhë janë: C/C++ (përmes Visual C++), VB.NET (përmes Visual Basic .NET), C# (përmes Visual C#) dhe F# (vetëm në Visual Studio 2010). Mbështetë edhe gjuhë te tjera siç janë M, Python dhe Ruby poashtu mbështetë XML/XSLT, HTML/XHTML, JavaScript dhe CSS.

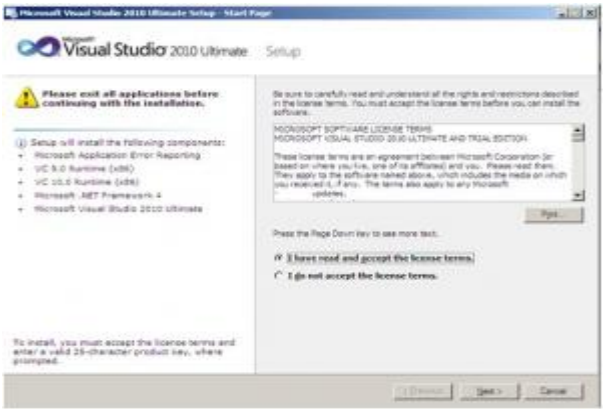
Microsoft na sjellë "Express" editionin të Visual Studio 2010 me gjuhët programuese Visual Basic, Visual C#, Visual C++ dhe Visual Web Developer free. Visual Studio 2010, 2008 dhe 2005 edicionin profesional, se bashku me versionet e gjuhëve specifike (Visual Basic, C++, C#, J#) të Visual Studio 2005 janë në dispozicion free për studentet përmes programit Microsoft DreamSparks.

Instalimi

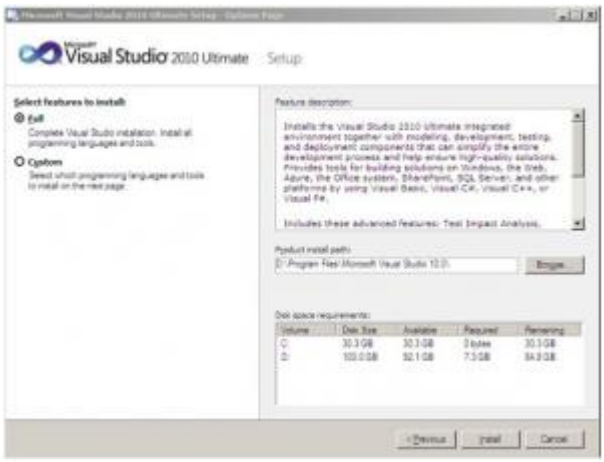
Në fillim e shkarkojmë nga interneti një version të softuerit Visual Studio (p.sh., Visual Studio 2008 Service Pack 1 "Shkarko nga cnet.com" ose Visual Studio 2010 Professional "Shkarko nga cnet.com").

Figure 1 - Setup-i është duke ngarkuar componentët e instalimit

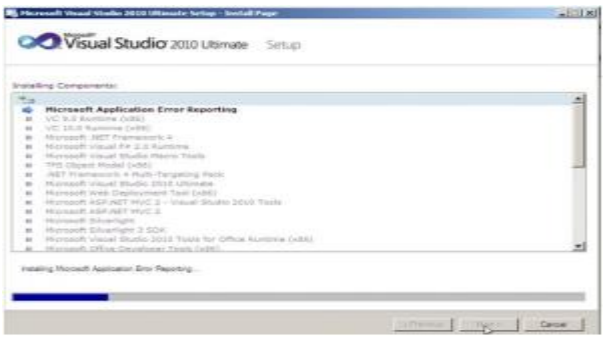
Pasi të kemi shkarkuar Visual Studion hapim setup-in përkatës, dhe në ekran do të na shfaqet figura e më poshtme:



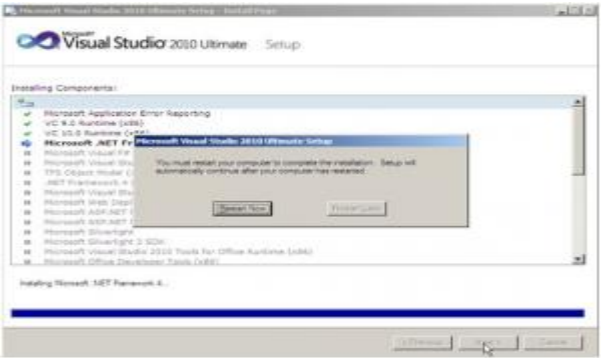
Tani duhet të pranojmë argumentet e licencës për të vazhduar instalimin. Pasi pranojmë shtypim **Next>>.**



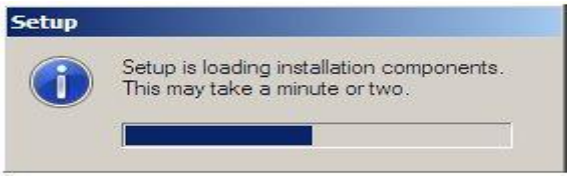
Tani zgjedhim karakteristikat që ne duam të instalohen dhe vendin ku duam ta instalojmë në hard disk. Pastaj klikojmë **Install.**



Setup-i instalon komponentët përkatëse.

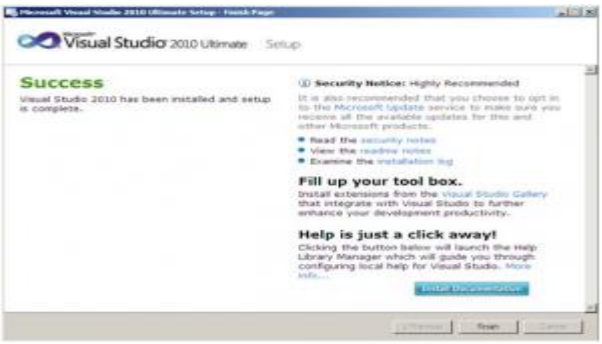


Setup-i kërkon qe sistemi të ristartohet pas instalimit të .Net Framework 4.0



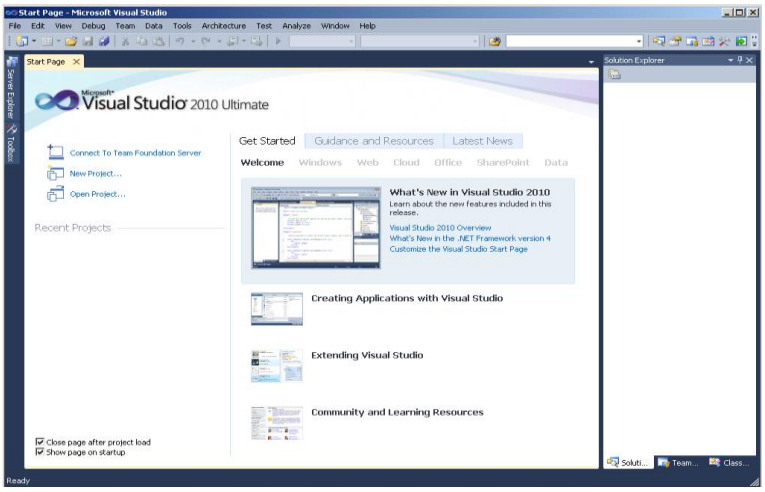
Pas restartimit setup-i starton përsëri.

Setup-i fillon të instalojë komponentët e ngelura.



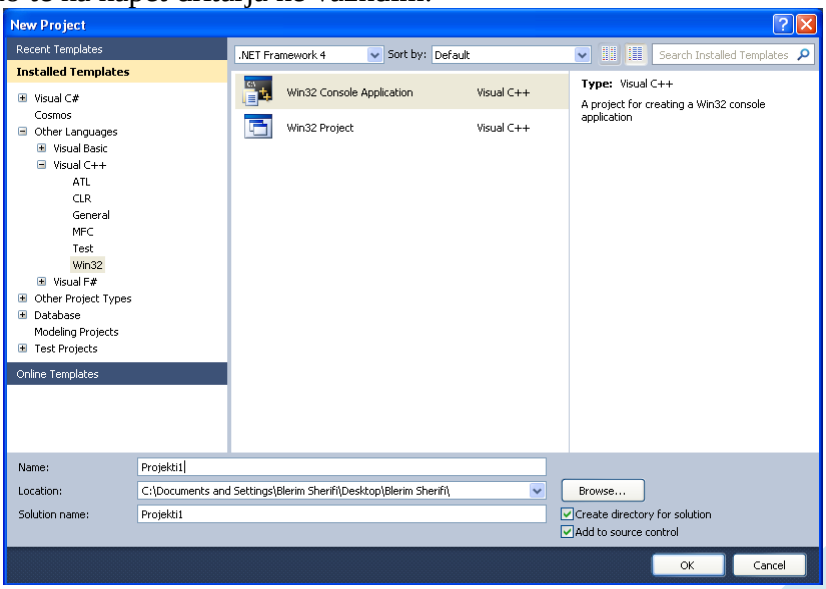
Instalimi u kompletua.

Pas instalimit të Vusual Studios e hapim atë dhe në ekran do të na paraqitet faqja e parë fillestare e VS Ultimate 2010.

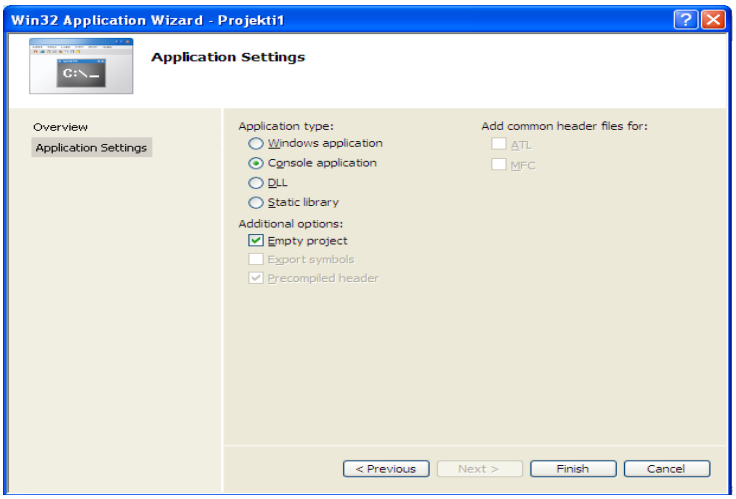


Përdorimi i Visual C++

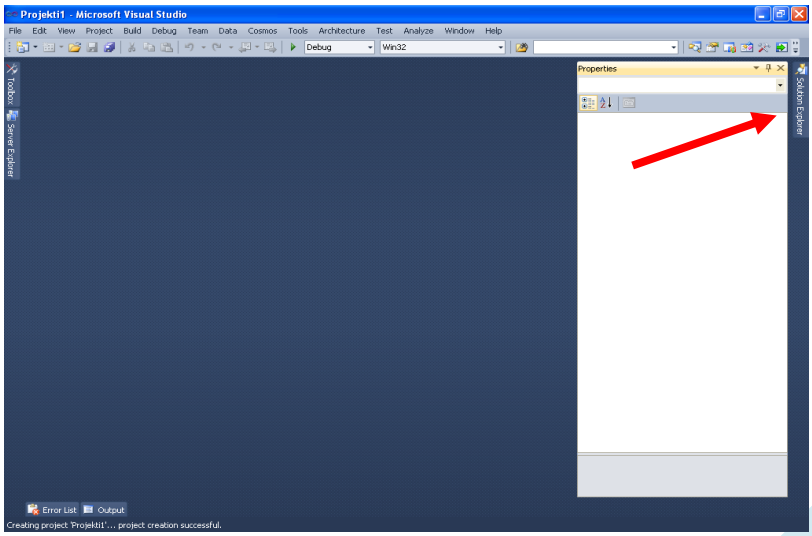
Për të shkruar një program në C++ në Visual Studio ne do të përdorim pjesën e konzolës. Pasi të kemi hapur VS, shkojmë tek menyja rënëse **File > New > Project** (ose kombinimin *Ctrl+Shift+N* nga tastiera) dhe do të na hapet dritarja në vazhdim:



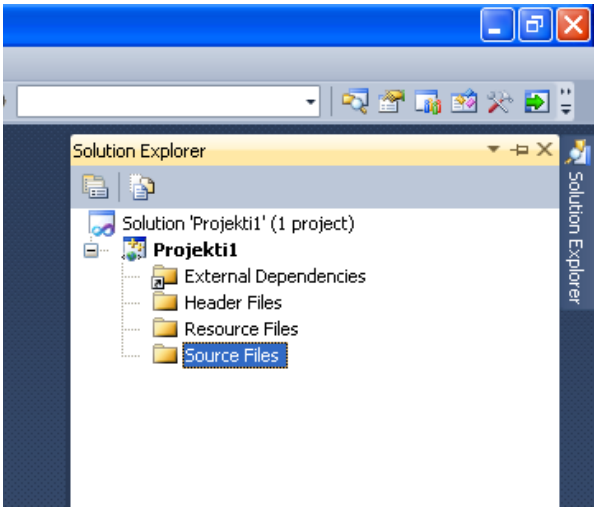
Tani tek ana e majtë kemi opsionin **Install Templates** ku zgjedhim nën opsionin **Other Languages** dhe **Visual C++**, ku selektojmë **Win32** dhe do të na paraqiten dy mundësi për të zgjedhur *Win32 Console Applciation* dhe *Win32 Project*. Ne do të zgjedhim të parën dhe tek fusha *Name* do të shënojmë emrin **"Projekti1"** dhe klikojmë **OK** (vërejtje, po ashtu kemi mundësi që tek butoni **Browse...** të zgjedhim lokacionin ku duam ta ruajmë projektin tonë). Tani kemi këtë dritare:



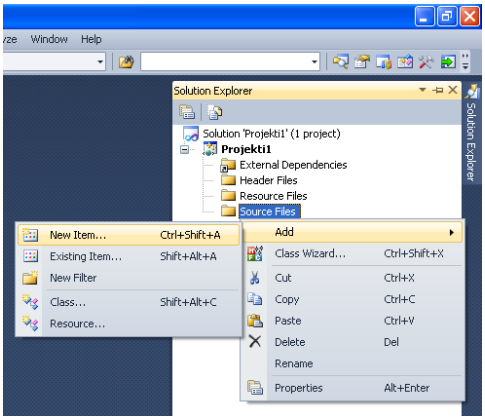
Në anën e majtë zgjedhim opsionin **Application Settings** pastaj aktivizojmë **Empty Project** dhe klikojmë **Finish** ku do të na paraqitet hapësira punuese e projektit të krijuar.



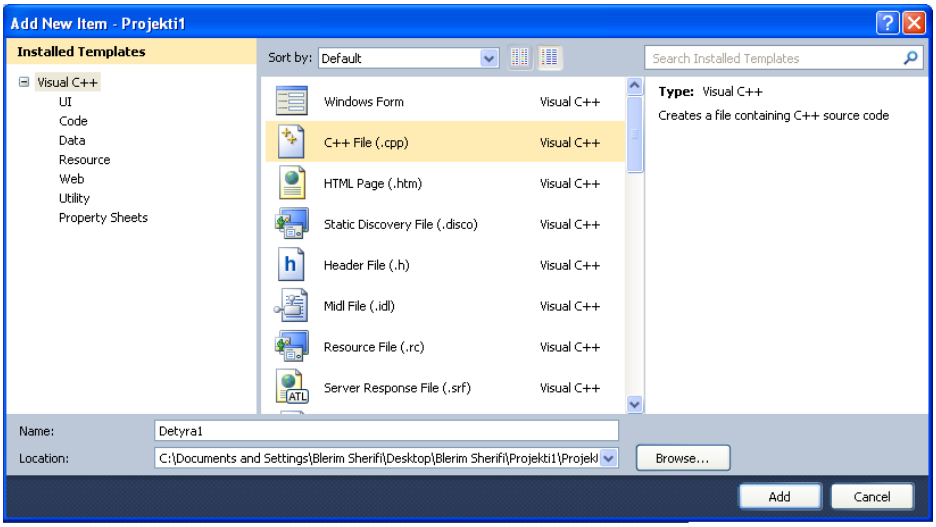
Tash tek ana e djathtë kemi dritaren **Solution Explorer** selektojmë atë dhe do të kemi:



Tek folderi **Soruce Files** klikojmë me të djathtën e miut dje zgjedhim opsionin **Add** dhe më pas **New Item...** (ose kombinimin Ctrl+Shift+A nga tastiera) siç shihet në figurën e mëposhtme:

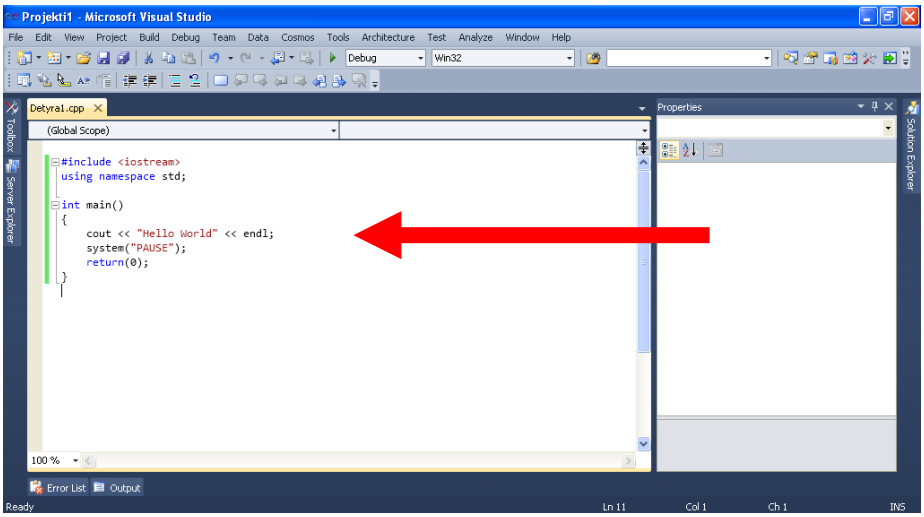


Pas këtij hapit kemi këtë dritare në vazhdim:



Tani zgjedhim **C++ File (.cpp)** dhe tek fusha **Name** shënojmë *Detyra1* dhe klikojmë **Add**.

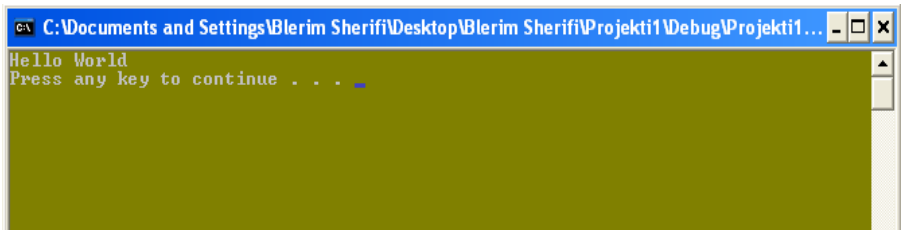
Pas këtij hapi në dritaren e Visual Studios do të na paraqitet hapsira punuese ku ne mund të shkruajmë kodin në C++, kjo hapësirë duket kështu:



Në këtë hapësirë shënojmë kodin në vazhdim:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello World" << endl;
    system("PAUSE");
    return(0);
}
```

Pasi të kemi shkruar këtë kod ekzekutojmë programin duke klikuar tek menyja rënëse **Debug>Start Debugging**. Dhe programi do të ekzekutohet dhe në ekran do të na shfaqet teksti *"Hello World"* kështu:



Kështu njëjtë do të shënohen të gjitha programet në vazhdim nëse përdorim Visual Studion.

Kapitulli III

B a z i k e t

Nënkapitujt:

- ✓ **Elementet e gjuhës C++**
- ✓ **Programi i parë në C++**
- ✓ **Biblioteka *Stream***
- ✓ **Identifikatorët (emrat)**
- ✓ **Variablat (ndryshoret)**
- ✓ **Komentet**
- ✓ **Operatorët *cout* dhe *cin***
- ✓ **Të dhënat standarde**
- ✓ **Memoria**
- ✓ **Stringjet**
- ✓ **Shembuj**
- ✓ **Kontrollim njohurishë**

Elementet e gjuhës C++

Para se të jepen njohuri të përgjithshme mbi rregullat e shkrimit të programeve në gjuhën C++, duhet të përcaktohen elementet e kësaj gjuhe, siç janë: të dhënat, identifikatorët, konstantet, variablat, pointerët, operatorët dhe shprehjet aritmetikore. Në këtë kapitull janë përfshirë elementet bazike të gjuhës C++.

Programi i pare ne C++

Për ekzekutimin e kodit na duhet një aplikacion. Si aplikacion më i njohur është Visual Studio paketë e kompanisë së njohur Microsoft. Por për lehtësim ne do të përdorim programin Code::Blocks që në përdorim është më i thjeshtë se Vizual Studio. Këtë program mundeni ta gjeni si burim të lirë në internet për shembull në këtë link: http://www.codeblocks.org/downloads . Përdorimi i këtij programi është sqaruar në kapitullin 2.

Le të krijojmë një program të thjeshtë i cili në ekran do të na nxjerr tekstin "**Programi im i pare**".

Programi nr 1:

```
1    #include <iostream.h>

2    int main ()
3    {
4    cout <<"Programi i parë";
5    }
```

Pra, ky program është i thjeshtë në të cilin gjëja që ne vet e krijojmë është teksti "**Programi im i pare**". Qe do te thotë se të gjithë pjesët tjera janë rregulla të programit.

Shënim:

Reshti I: Kjo linjë përdor direktivën prepocesorike **#include** për të përfshirë përmbajtjen e header fajllit **iostream.h** në program. Iostream.h është një header fajll standard C++ dhe përmban përkufizime për hyrje dhe dalje.

Reshti II: Kjo linjë përcakton funksionin të quajtur **"funksioni kryesorë"**. Një funksion mund të ketë zero ose më shumë **parametra**; këto gjithmonë shfaqen pas emrit të funksionit në mes të një palë kllapave gjarpëruese "{ }", ku kllapa e parë tregon fillimin e trupit **main**, kurse e dyta tregon mbarimin e trupit **mai**. Fjala **void** e shfaqur në mes të kllapave tregon se **main** nuk ka parametra. Një funksion mund të ketë tip kthimi (**return type**); kjo gjithmonë shfaqet para emrit të funksionit. Lloji i kthimit të funksionit kryesorë **main** është **int** (dmth. një numër i plotë). Të gjithë C++ programet patjetër të kenë saktësisht një funksion kryesor **main**.

Reshti III: Kjo kllapë gjarpëruese tregon fillimin e trupit **main**.

Reshti IV: Kjo linjë është një **deklarimi** (**statemant**). Deklarimi është një hap llogaritje që mund të prodhoj një vlerë. Në mbarim të çdo deklarimi shënohet gjithmonë pikëpresje (;). Ky deklarim bën që stringu "Programi im i parë" të dërgohet në **cout** rrjedhën e daljes. Një varg apo string është çdo sekuencë e karaktereve të mbyllura në dy citate ("thojësz")

Reshti V: Kjo kllapë e pasme gjarpëruese tregon mbarimin e trupit **main**.

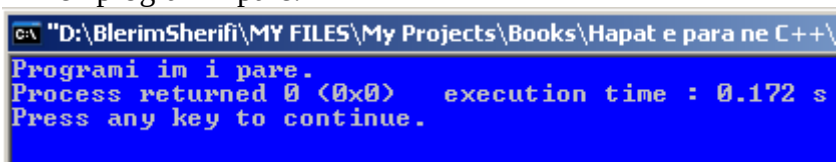
Pra nga kjo analizë ne kuptojmë thelbin e shkrimit të një programi në C++, pra, më konkretisht njihemi me rregullat e shkrimit të programeve në këtë gjuhë. Siç u tha edhe më lartë gjëja që ne mundemi ta ndryshojmë në këtë program është teksti **"Programi im i pare"**, ku në vazhdim do ta zëvendësojmë me tekstin **"Programori fillestar"**.

Programi nr 2:

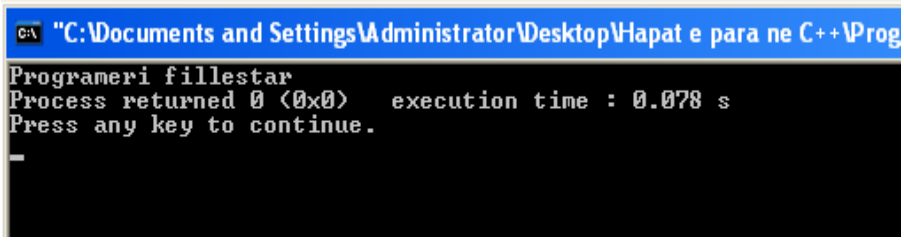
```
# include <iostream>

using namespace std;   // ky resht përdoret për të njohur cin dhe cout

int main ();
{
        cout << "Programeri fillestar";
        return 0;
}
```

Dalja në ekran pas ekzekutimit të këtyre programeve me ndonjë program të veçantë (p.sh. Code::Blocks, Visual Studio, DevC++, etj.), do të ishte:

Tek programi i pare:



Tek programi i dyte:



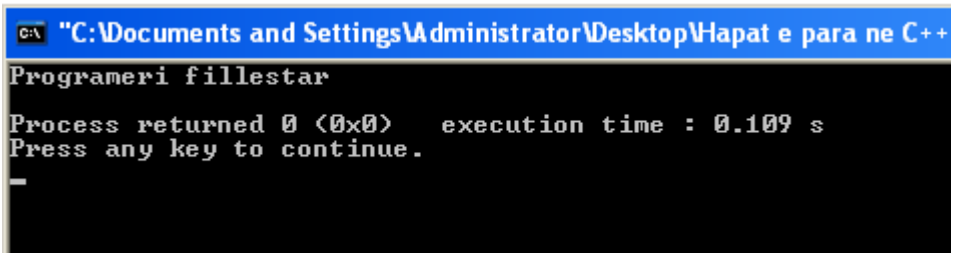
Ky program fillestar mund që të shkruhet edhe ndryshe por të jap të njëjtin rezultat si më parë.

Programi nr 3:

```
# include <iostream>
using namespace std;
int main ();
{
cout << "Programeri fillestar" << endl;

return 0;
```

Ndryshimi që është bërë në këtë program është shtesa **endl,** kjo shtesë kalon kursorin në resht të ri, pra është komandë për kalim në resht të rijë.

Gjatë ekzekutimit të këtij programi në ekran do të kemi:



Programi mund të shënohet edhe në këtë formë:

Programi nr 4:

```
# include <iostream>
using namespace std;
int main ();
{
cout << "Programeri fillestar"
     << endl;
return 0;
```

Edhe përkundër këtij ndryshimi dalja në ekran do të jetë e njëjtë, vetëm se ky ndryshim bën që programi jonë të duket më i qartë e i bukur.

Biblioteka stream

Biblioteka e quajtur stream është përkufizuar për përdorim me C++, me qëllim të ngritjes së efektivitetit të gjuhës. Kjo bibliotekë u mundëson programuesve të drejtën mbi të gjitha formatimet e mundshme, duke kërkuar vetë pjesë të mundshme.

Identifikatorët (Emrat)

Njësitë elementare memoruese në të cilat vendosen të dhënat dhe rezultatet e programeve emërohen duke përdorur *identifikatorë* (ang. identifier). Kështu, identifikatorët përdoren për emrat e konstanteve, variablave, nënprogrameve dhe strukturave të tjera të përfshira në program. Identifikatorët formohen si kombinim i *shkronjave* (a, b, ..., z, A, B, ..., Z), *numrave* (0, 1, 2, ..., 9) dhe *nënviza* (_). Simboli i parë në

identifikatorë mund të jetë shkronjë ose nënvijë. Kështu, p.sh., si identifikatorë mund të merren:

numri
vlera
distanca_mes_reshtave
EmriiProgramit
_Fundi
A4C7

Nuk lejohet që simboli i parë në identifikatorë të jetë numër, ose identifikatori të përmbajë simbole speciale (simbole që nuk janë shkronja ose numra), p.sh., siç janë: !, $, %, +, > etj. Ja disa identifikatorë të gabueshëm që programi nug do ti njohë apo do ti njohë gabim:

8Shtatori	// gabim pasi fillon me numër
Pesë+Katër	// gabim pasi ka +
Sony Eriksson	// gabim pasi ka hapsirë mes fjalëve
US$	// gabim pasi ka shenjën $
%fitimi	// gabim pasi ka %
Dolce&Gabbana	// gabim pasi ka &.

Gjithashtu dua të cek se shkronjat e alfabetit shqip ë, Ë, ç dhe Ç nuk mund të përdoren gjatë formimit të identifikatorëve.

Në gjuhën C++ gjatësia e identifikatorëve nuk është e kufizuar. Por, përdorimi i identifikatorëve të gjatë është jopraktik e ndonjëherë edhe me probleme gjatë kompajlimit. Fjalët kyçe të cilat përdoren në gjuhën C++ (fjalët e rezervuara nga gjuha C++), nuk lejohet të përdoren si identifikatorë. Kështu, p.sh., si identifikatorë nuk mund të përdoren fjalët:

else
cout
if
while
string
do
goto
int

double
char
float

sepse janë fjalë të rezervuara nga gjuha C++.

Kombinimet që fillojnë me simbolin për nënvizim në gjuhën C++ kanë një përdorim të veçantë, për këtë arsye është e preferueshme që gjatë formimit të identifikatorëve të zakonshëm këto kombinime të mos përdoren. Por, simboli për nënvizim mund të përdoret si lidhës gjatë formimit të identifikatorëve më të gjatë, p.sh., kështu:

koha_e_bukur
rrezet_e_arta_te_diellit
shuma_me_e_madhe

Praktikohet edhe kjo formë e formimit të identifikatorëve të dhënë në shembullin e mësipërm:

kohaebukur
rrezetediellit
shumameemadhe

ose edhe forma:

KohaeBukur
RrezetEArtaTeDiellit
ShumaMeEMadhe

Kompajleri i gjuhës C++ gjatë formimit të identifikatorëve **i dallon shkronjat e vogla dhe shkronjat e mëdha**. Kjo, d.m.th. se, p.sh., nuk merren si identifikatorë të njëjtë kombinimet:

Nata
nata
DITA
ditA

Me qëllim të evitimit të gabimeve të mundshme për shkak të dallimit në madhësi të shkronjave, gjatë programimit në gjuhën C++ kryesisht shfrytëzohen shkronjat e vogla.

Variablat

Të dhënat dhe rezultatet që fitohen gjatë llogaritjeve të ndryshme ruhen në hapësirën e memories së kompjuterit si vargje të shifrave binare. Meqë gjatë ekzekutimit të programit në kompjuter, në një lokacion të memories mund të vendosen vlera të ndryshme, përkatësisht vlerat brenda tyre mund të jenë variabile, lokacionet e memories paraqesin *variabla* (ang. variable). Përmbajtja e një lokacioni njihet edhe si *vlerë e variablës*, kurse emri simbolik që i shoqërohet e paraqet *identifikatorin e variablës*, ose *emrin e variablës* (ang. variable name).

Të gjithë variablat kanë dy atribute të rëndësishme:

- **Tipi** që është i vendosur kur variabla është definuar (p.sh. numer i plote, real, karakter). Pasi të jetë përcaktuar, lloji i variablës nuk mund të ndryshohet.
- **Vlera** e cila mund të ndryshohet duke i caktuar një vlerë të re ndryshores. Llojin e vlerave variabla mund ta supozojë varësisht nga tipi i sajë. Për shembull një variabël e tipit integer mund të merr vetëm vlerë të numrave të plotë (p.sh. 10, -5, 500).

Shuembull:

```
1   #include <iostream.h>
2   using namespace std;

3   int main()
4   {
5   int numri1;       //deklarimi i numrit te pare të tipit integer (të plotë)
6   double numri2;  //deklarimi i numrit të dytë të tipit double (real)

7   numri1 = 10;
8   numri2 = 5.5;

9   cout <<"Numri i pare i tipit int: "
          << numri1
        << endl;
10  cout <<"Numri i dyte i tipit double: "
          << numri2
          << endl;
11  return 0;
12  }
```

Reshti V: Kjo linjë përcakton një variabël të tipit *int* (integer) me emrin **numri1.**

ReshtiVI: Ndërsa kjo linjë përcakton një variabël të tipit double (rela) me emrin **numri2**.

Reshti VII: Tek kjo linjë variablës numri1 i shoqërohet vlera 10.

Reshti VIII: Ndërsa te kjo linjë variablës **numri2** i shoqërohet vlera reale **5.5**.

Dalja në ekran do të jetë:

```
Numri i pare i tipit int: 10
Numri i dyte i tipit double: 5.5

Process returned 0 (0x0)   execution time : 1.910 s
Press any key to continue.
```

Deklarimi i variablave të zakonshme

Për çdo variabël, para se të shfrytëzohet, duhet të deklarohet tipi i saj. Ky deklarim, p.sh., bëhet kështu:

```
int a;
double b;
float sh;
short int koha;
char z;
string emri;
```

Brenda një deklarimi mund të përfshihen edhe më shumë variabla, duke i ndarë me presje, p.sh., kështu:

```
int x,y;
long int dita,nata,a3;
```

Më shumë deklarime të variablave mund të shkruhen në një rresht, p.sh., kështu:

```
double a,b;  int z;  float l,o;
```

ku, siç shihet, deklarimet e tipave të veçanta janë ndarë mes vete me pikëpresje (**;**).

Deklarimi i tipit të njëjtë brenda një programi mund të paraqitet edhe më shumë
herë, p.sh., kështu:

```
int i,j;
int s;
```

gjë që mund të bëhet edhe më shkurt në këtë mënyrë:

```
int i,j,s;
```

Shembull:

```
#include <iostream>
using namespace std;
int main()
{
   int a;
   double b;
   char c;
   float d;
   short int e;

   a = 5004;      b = 48.505;      c = 'e';      d = 1.5;       e = 7;

   cout <<"a= "<<a
       <<"\nb= "<<b
       <<"\nc= "<<c
       <<"\nd= "<<d
       <<"\ne= "<<e
       <<endl;

  return 0;
}
```

Dalja në ekran pas ekzekutimit:

```
a= 5004
b= 48.505
c= e
d= 1.5
e= 7

Process returned 0 (0x0)   execution time : 0.640 s
Press any key to continue.
```

Komentet

Që programi të jetë i kuptueshëm, qoftë edhe pas një kohe më të gjatë, ose edhe nga shfrytëzues të tjerë, në pjesë të ndryshme të tij mund të shkruhen tekste, të cilat njihen edhe si *komente* (ang. comment).

Në gjuhën C++ komentet mund të shkruhen në dy mënyra:

komente brenda një ***rreshti*** (ang. end-of-line comment) dhe

komente brenda një ***blloku*** (ang. block comment).

Komentet brenda një rreshti fillojnë me dy vija të pjerrëta //, diku pas komandës së shkruar në atë rresht dhe vazhdojnë deri në fund të rreshtit.

Shpesh për komente nevojiten tekste më të gjata. Për këtë qëllim komentet shkruhen brenda bllokut i cili fillon me **/*** dhe përfundon me ***/**. Në këtë mënyrë mund të shkruhen edhe komentet brenda një rreshti.

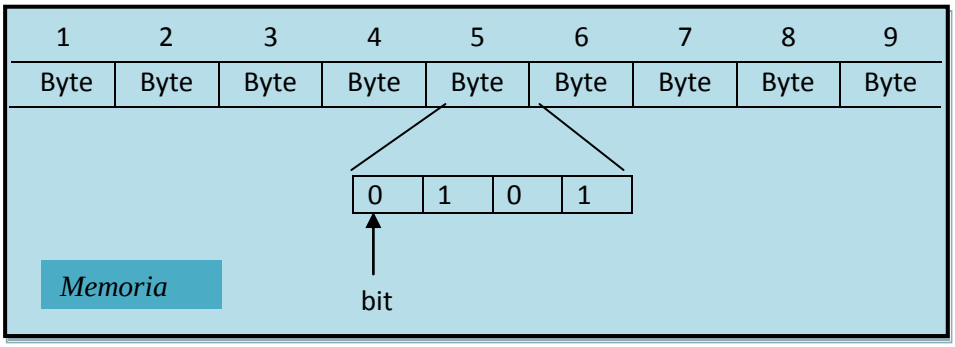
Gjatë kompajlimit të programit, kompjuteri komentet i eliminon, sepse ato shfrytëzohen vetëm nga përpiluesi i programit. Po ashtu komentet në Code::Block janë me ngjyrë të hijezuar;

Shembull:

```
#include <iostream>
using namespace std;
int main ();
{
        // Ky eshte komen dhe nuk del në ekran
        cout << "Mire se erdhe ne C++!";
        /* _____________________________________
        |       ketu eshte nje bllok            |
        |       ku mund te shkruhet me shum     |
        |       koment                          |
        |_______________________________________|*/
        return 0;
}
```

Memoria

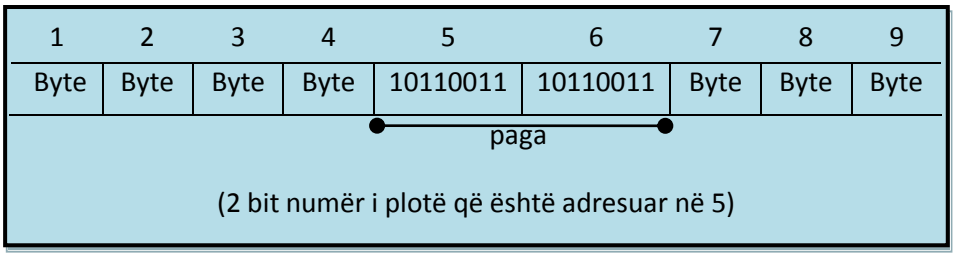
Çdo kompjuter ka memorien për punë të përkohshme RAM (Readom Acces Memory), në të cilën ruhen programet kur ne i ekzekutojmë ato. Kjo memorie mund të mendohet si një sekuencë e afër e bitëve, ku secila prej këtyre bitëve është e aftë për të ruajtur shifrën binare (0 ose 1). Bitët janë të adresuar në mënyrë të njëpasnjëshme (sekuenciale). Prandaj se cili bit mund të identifikohet në mënyrë të vetme (unike) me adresën përkatëse (shif figurë më poshtë).



Kompajleri në këtë rast gjeneron kodin ekzekutues i cili shënon entitetet (subjektet) e të dhënave në lokacionet e memories. Për shembull, deklarimi i variablës

int paga = 65000;

bën që kompajleri të rezervon (ndan) pak bajt për të pasqyruar *pagën*. Numri i saktë i bitëve të ndara dhe metoda për përfaqësimin binar të numrave të plotë varet nga specifikat e implementimit të gjuhës C++. Por le të themi dy bit të koduar si 2's integer. Kompajleri përdor adresën e bitit të parë me të cilën paga është ndarë për t'iu referuar asaj. Caktimi i më sipërm bën që vlera 65000 të ruhet si 2's kompliment integer në dy bit të ndara (rezervuara) (shih figurën poshtë).



Përderisa prezantimi i saktë binarë i një artikulli të dhënash është i rrallë në interes të një programuesi, organizimi i përgjithshëm i memories dhe përdorimi i adresave për referim tek e dhëna është shumë i rëndësishëm.

Operatorët *cout* dhe *cin*

Këto dy operatërë përdoren në të gjitha programet që mund të ndërtohen në C++.

Operatori **cout** përdoret për dalje në ekran të teksteve dhe numrave. Ky operator shkruhet në këtë formë të tijë fillestare:

Për nxjerjen e teksteve **cout** përdoret në këtë formë :
cout << " ... ";
Shenja dy herë më e vogël "<<" është e patjetërsueshme kur përdoret **cout**, ndërsa në apostrofa që janë të patjetërsueshme në këtë rast, shënohet teksti që dëshirojmë të del në ekran. Pikë presja "**;**" vendohet në fund të çdo reshti kur ai përfundon. Në qoftë se nuk e shënojmë ndonjërën shenjë që janë të patjetërsueshme në C++, për shembull nxjerrjen e një teksti pa apostrofa programi do të na nxjerr gabime në kod (**Erorr**).

```
cout << Programi im i pare;          // Gabim
cout << "Programi im i pare";        // Saktë
```

Gjithashtu programi do të na nxjer gabim edhe nëse harojmë të vëndojmë pikë presje "**;**" në fund të rreshtit ose harrojmë shenjën "<<" pas **cout**-it.
Për nxjerrjen e variablave **cout** ndryshon dhe do të duket:

```
cout << variabla ;
```

Pra siç shohim këtu, variablat nuk shënohen në apostrofa siç shënoheshin tekstet. Për shembull nëse në program deklarojmë një variabël me emrin **numri** e cila e ka vleren 5, dhe përmes **cout** deshirojmë ta nxjerim në ekran do të kemi:

cout << numri;

Dhe në ekran nuk do të na paraqitet teksti **numri** pasi ai është emri i variablës të deklaruar më parë, por do të na paraqitet vlera e saj, pra numri 5.

Operatori **cin** përdoret për futjen e të dhënave nga tastiera në program. Pra përmes **cin** ne mundemi të fusim të dhëna të ndryshme në program qofshin tekste ose numra. Forma fillestare e **cin** është:

cin >> variabla;

Këtu shohim se **cin** ndryshon nga **cout** përmes shenjës më e madhe ">>", kurse në vend të tre pikave shënojmë variablën e cila do ta merr vlerën që ne dëshirojmë ta fusim. Nëse dëshirojmë të fusim karakter për shembull shkronjën **a** atëherë variabla e deklaruar do të merr vlerën **a** dhe në ekran do të del si **a**, kurse nëse duam të fusim numër për shembull numrin **1** në program variabla e deklaruar do të merr vlerën **1**, pra në ekran do të del numri **1**.

Të dhënat standarde

Informatat të cilat i jepen kompjuterit në një formë të kuptueshme për të, quhen *të dhëna* (ang. data). Varësisht nga natyra dhe kufijtë e vlerave të tyre, në gjuhën programuese C++ shfrytëzohen tip të ndryshme tët dhënave. Në kolonën e parë të *Tabelës1.1* janë numëruar të dhënat standarde që shfrytëzohen në gjuhën C++.

Të dhënat standarde në C++		
Tipi	**Kufiri i vlerave**	**Bajtë**
short **short int**	-32768...32767	2
int	-2147483648... 2147483647	4
long **long int**	-2147483648... 2147483647	4

unsigned short **unsigned short int**	0...65535	2
unsigned **unsigned int**	0...4294967295	4
unsigned long **unsigned long int**	0...4294967295	4
float	1.2e-38...3.4e+38	4
double	1.7e-308...1.7e+308	8
long double	3.4e-4932...1.1e+4932	10
char	-128...127	1
unsigned char	0...255	1
signed char	-128...127	1
wchar_t	0...65535	2
bool	True, false	1

Tabela 1.1 – Të dhënat standarde në C++

Numrat e plotë

Tipet e të dhënave të cilët paraqiten me **short**, **short int**, **int**, **long** dhe **long int** u përkasin numrave të plotë.

Për shembull një variabël e deklaruar me **int** përpara, vlerat e mundshme për të janë -2147483648 deri në 214783647. Për tipat tjera tët dhënave shiko *tabelën 1*.

Numrat natyrorë

Tipat e të dhënave që u përkasin numrave natyrorë, përfshirë këtu edhe zeron paraqiten me shtesën unsignet përpara. Pra kemi **unsignet short**, **unsignet short int**, **unsigned, unsigned int, unsigned long** dhe **unsigned long int**.

Unsigned short dhe **unsigned short int** marrin vleren prej **0** deri **65535**. Për tipet tjera të të dhënave shiko *tabelën 1*.

Numrat e dhjetorë

Tipat e të dhënave që u përkasin numrave dhjetorë janë: **float, double** dhe **long double.** Këto numra ndryshe quhen edhe numra real. Vlerat që mund ti marrin këto tipe të të dhënave janë shënuar në *tabelën 1*.

Të dhënat karakter

Tipat e të dhënave që u përkasin karaktereve janë: **char, unsigned char** dhe **signed char**. Këto tipa tët dhënave mund që të ruajnë karaktere dhe simbole të ndryshme. Kufiri i vlerave të mundshme për këto tipa të dhënash mund që ti shikoni në *tabelën 1*. Një variabël karakter zë një bajt të vetëm ku regjistrohet kodi i karakterit. Ky kod është një vlerë numerike dhe varet nga *kodimi i sistemit të karaktereve* që përdoret. Sistemi më i zakonshëm i kodimit të karaktereve është ai ASCII (American Standard Code for Information Interchange). Për shembull shkronja A ka ASCII kodin 65, kurse shkronja e vogël a ka kodin ASCII 97.

Deklarimi i të dhënave të tipit karakter:

```
char karakteri = 'A';
```

Siç shihet edhe nga deklarimi vlera e karakterit shënohet në kuota të njëfishta (p.sh. 'A').

signed char – variablat e këtij tipi mund të përmbajnë numra nga -128 deri 127.

unsigned char – variablat e këtij tipi mund të përmbajnë numra nga 0 deri 255.

Një shembull praktik:

```
signed char fillimi = -100;
unsigned char  reshti = 4, kolona = 10;
```

Gjithashtu kemi edhe karaktere të cilat i shënojmë por që nuk shtypen, në fakt këto prezantojnë sekuenca të caktuara. Për shembull:

```
'\n'    // resht të ri
'\r'    // kthim të qëndrimit
'\t'    // tab horizontal (8 vende të lira)
'\v'    // tab vertikal
'\b'    // kthim mbrapa (backspace)
'\f'    // formfeed
'\a'    // zë (alert).
```

Por në qoftë se duam të nxjerrim shkruajmë një tekst brenda kuotave të dyfishta atë nuk mund ta bëjmë në mënyrë të thjeshtë por duhet patjetër të përdorim këtë formë:

```
‘ \” ’          // printon kuota të dyfishta në tekst,
‘ \’ ‘          // printon kuota të njëfishta në tekst,
‘ \\ ‘          // printon një vijë të pjerrtë ( \ ).
```

Këto karaktere gjithashtu mund të caktohen duke përdorur vlerën e tyre të koduar numerike.

Stringjet

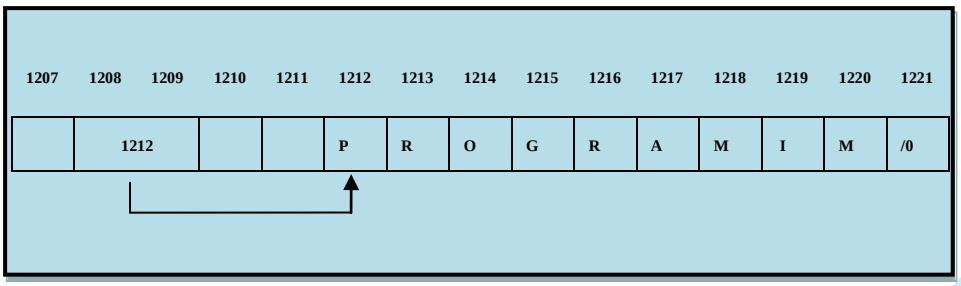
Vargjet e simboleve të ndryshëm (shkronjave, numrave dhe simboleve speciale), përkatësisht tekstet e çfarëdoshme të shkruara brenda thonjëzave, quhen *stringje* dhe llogariten si të dhëna të tipit **string** .

Të dhëna të tipit string mund të jenë:

“Programi im i pare”
“Programori fillestar”
“x=3*(4+2)”

Këto *strungje* formohen si vargje të karaktereve andaj ato njihen edhe si stringje të karaktereve, ku kompjuteri në fund ua shton edhe karakterin zero (**null**) ‘**\0**’, për ta përcaktuar fundin e tyre.

Kështu stringu **“PROGRAMIM”** në memorien e kompjuterit vendoset në këtë mënyrë:



duke shfrytëzuar nga një bajt për çdo karakter, ndërsa deklarimi i këtij stringu bëhet kështu:

```
string a;                    //deklarimi
a = "PROGRAMIM"              //shoqërimi i vlerës
cout << a;                   //dalja në ekran
```

siç shihet në figurën e lartë shënuar, stringu në memorie ruhet në atë mënyrë që karakteret brenda stringut regjistrohen në bit të veçanta ku një karakter i referohet një biti dhe në fund të stringut shënohet **null karakteri \0**. Kjo shenjë i jep të kuptoj kompjuterit se stingu përkatës aty përfundon. Kështu nuk mund të ketë bit të lirë (të zbrazët) mes karaktereve pasi kompjuteri nuk do të dijë se ku përfundon stringu, prandaj në gjuhën angleze quhen *literal string*, që në përkthim do të thotë string i një pas njëshëm. Ndërsa në qelinë me numër 1208 dhe 1209 është regjistruar referenca e stringut, pra fillimi i stringut, kështu që ne kur dëshirojmë të përdorim këtë string kompajleri merr referencën e stringut dhe kalon që aty pra tek qelia me numër 1212 ku fillon leximin e stringut një pas një dhe i nxjerr ato në ekran. Për shembull tek rasti jonë kur ne shënojmë komandën "**cout << a;**" kompjuteri do të merr referencën e ruajtur në qelinë numër 1208 dhe 1209 (që llogaritet si një qeli) dhe kalon tek qelia 1212 dhe fillon nxjerrjen në ekran të tekstit **"PROGRAMIM"** shkronjë për shkronje me radhë deri sa të arrijë tek karakteri **null \0** ku tregon se përfundon stringu. Dhe në ekran do të del teksti **"PROGRAMIM"**.

Gjithashtu ka strigje të cilët janë të gjatë dhe kalojnë përtej një reshti, kështu stringjet e tilla shënohen me backslash kështu:

```
"Shembull për të treguar \
 se si të shënohen \
 stringjet e gjata me backslash."
```

Në këtë rast backslash përdoret për të treguar kompajlerit se stringu vazhdon në reshtin e ardhshëm. Stringu i mësipërm është i ngjashëm me këtë string në një resht të vetëm.

"Shembull për të treguar se si të shënohen stringjet e gjata me backslash."

Si zakonisht kur shënojmë një karakter të vetëm ndodh që ai të shënohet në kuota të dyfishta (p.sh. "A"), gjë që nuk është ekuivalente me

(p.sh. ‘A’). Në rastin e parë në memorie ndahen dy bit pasi në fund kemi edhe karakterin **null (\0)**, kurse tek rasti i dytë kemi vetëm një bit të ndarë.

Stringu më i shkurt i mundur është stringu **null**. Ky string shënohet me kuota të dyfishtë por të zbrazura (p.sh. **“”**), që thjeshtë nënkupton *karakterin null*.

Shembuj

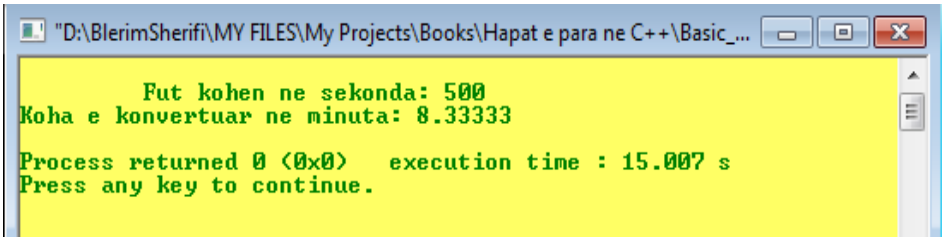
Në këtë pjesë do të ilustrojmë disa shembuj të zgjidhur dhe disa të pa zgjidhur për ti testuar njohuritë e mësuara deri tani.

Shembull1: Të shkruhet program që përdoruesi të fut nga tastiera kohën në sekonda ndërsa programi të nxjerr në dalje kohën e konvertuar në minuta.

Zgjidhja:

```
# include <iostream>
using namespace std;
int main ()
{
   // deklarimi i variables sekonda dhe minuta
   float sekonda;
   float minuta;
   // futja e kohes ne sekonda
   cout << "\n\t Fut kohen ne sekonda: ";
   cin >> sekonda;
  // konvertimi
   minuta = sekonda / 60;
   // nxjerja ne ekran
   cout << "Koha e konvertuar ne minuta: "
        << minuta
        << endl;
   return 0;
}
```

Dalja në ekran pas ekzekutimit do të duket si në figurën e më poshtme:

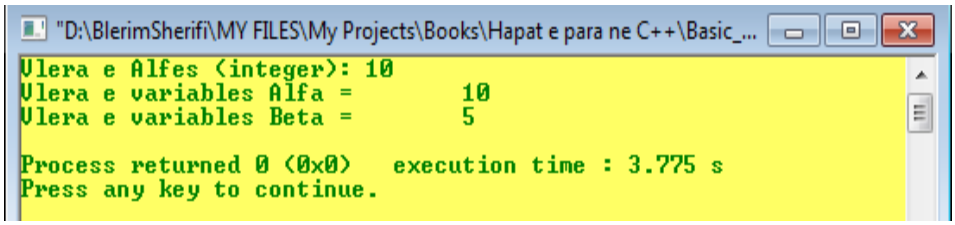


Shembull2: Të ndërtohet një program ku do të deklarohen dy variabla, njëra me emrin Alfa kurse tjetra Beta. Variabla Alfa të deklarohet e tipit int kurse variabla Beta të deklarohet e tipit double. Vlera e variablës Afla të futet përmes tastierës, kurse vlera e variablës beta të jetë sa dyfishi i variablës afla e shumëzuar me 0.5. Të dy variablat të shtypen në ekran.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   // deklarimi i variablave
   int Alfa;
   double Beta;

   // futja e vleres se Alfes nga tastiera
   cout << "Vlera e Alfes (integer): ";
   cin >> Alfa;

   // llogaritja e Betes
   Beta = Alfa * 0.5;

   // nxjerrja ne ekran
   cout << "Vlera e variables Alfa =\t" << Alfa
        << "\nVlera e variables Beta =\t" << Beta
        << endl;

return 0;
}
```

Sic shohim nga zgjidhja e detyrës, tek nxjerrja në ekran e variablave është përdorur shprehja **\t** e cila zhvendos kursorin në mënyrë horizontale për tetë vende të lira. Rezultati do to jetë i njëjtë si në figurën e më poshtme:



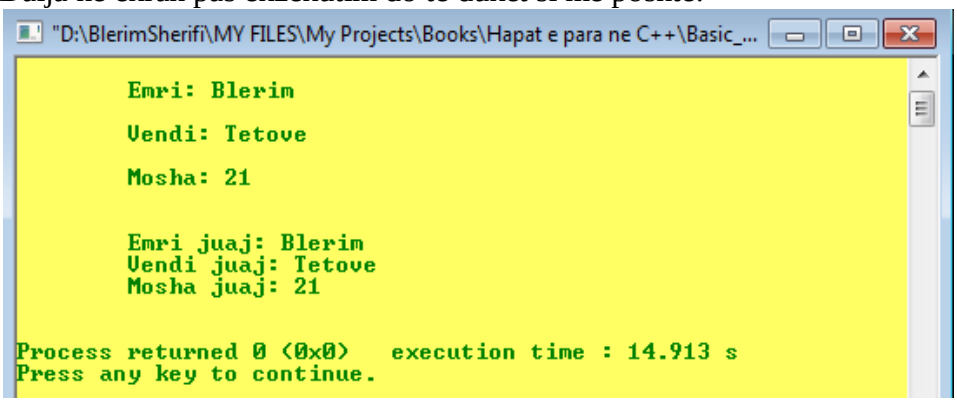
Shembull3: Të bëhet program në të cilin shfrytëzuesi të fut nga tastiera emrin, vendin dhe moshën e tij, dhe pastaj programi të nxjerr në ekran:

Emri juaj: emri.
Vendi juaj: vendi.
Mosha juaj: mosha.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   // deklarimi i variablave
   char emri[20];
   char vendi[15];
   int mosha;

   // leximi i vlerave nga tastiera
   cout <<"\n\tEmri: ";
   cin >> emri;
   cout <<"\n\tVendi: ";
   cin >>vendi;
   cout <<"\n\tMosha: ";
   cin >> mosha;

   // nxjerja ne ekran
   cout << "\n\n";
   cout << "\tEmri juaj: " << emri
        << "\n\tVendi juaj: " << vendi
        << "\n\tMosha juaj: " << mosha
        << "\n\n";
   return 0;
}
```

Këtu tek deklarimi i variablave emri dhe vendi janë të deklaruara të tipit char, pra karakter, kurse numri brenda kllapave të mëdha tregon se sa karaktere mund të mer variabla emri, pra në këtë rast 20 karaktere ([20]), kurse variabla vendi 15 karaktere ([15]). Gjithashtu tek ky shembull janë përdorur edhe shprehjet **\n** dhe **\t** që bëjnë programin tonë të duket më i qartë.
Dalja në ekran pas ekzekutim do të duket si më poshtë:

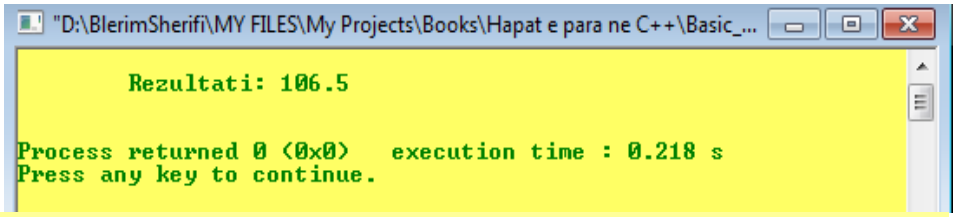


Shembull4: Të krijohet program që do të llogarit këtë formulë: 5*(10a-b)/2. Vlera e variablës a = 5, kurse b = 7.4. Rezultati të shfaqet në ekran.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   int a = 5;              // deklarimi i variablës a e tipit int dhe vlera 5
   double b = 7.4;         // deklarimi i variablës b e tipit double dhe vlera 7.4
   double rezultati;       //deklarimi i variablës rezultati

   rezultati = 5 * (10*a - b) / 2;   // llogaritja e formulës dhe ruajtja tek rezultati

   cout << "\n\tRezultati: "              // nxjerja ne ekran e rezultatit
      << rezultati
      << "\n\n";
   return 0;
}
```

Dalja në ekran do të jetë:

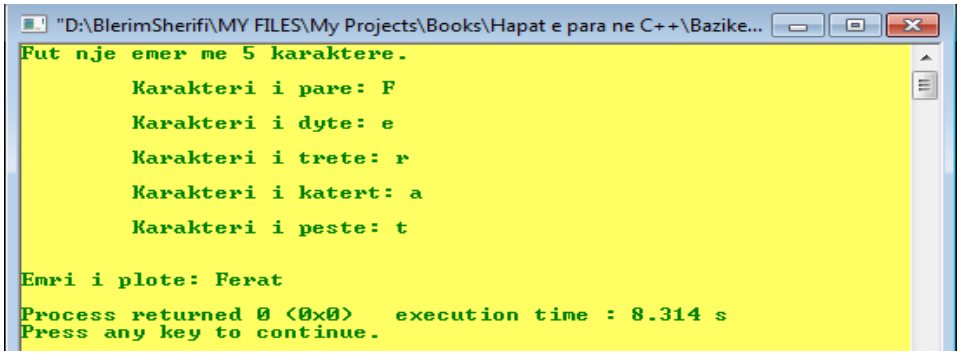


Shembull5: Të bëhet program që do të futet një emër nga tastiera shkronjë për shkronje. Në ekran të del emri i plotë.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   char k1, k2, k3, k4, k5;
   cout <<"Fut nje emer me 5 karaktere.\n";
   cout << "\n\tKarakteri i pare: "; cin >> k1;
   cout << "\n\tKarakteri i dyte: "; cin >> k2;
   cout << "\n\tKarakteri i trete: "; cin >> k3;
   cout << "\n\tKarakteri i katert: ";          cin >> k4;
   cout << "\n\tKarakteri i peste: " cin >> k5;
   cout << "\n\nEmri i plote: "
        << k1 << k2 << k3 << k4 << k5
        << endl;
   return 0;
}
```

Dalja në ekran:



Kontrollim njohurish

1. Të bëhet program që mbledh dy numra të futur nga tastiera dhe nxjerr në dalje shumën.
2. Cilët nga këto deklarime është i saktë?

   - int n = -100;
   - unsigned int i = -100;
   - signed int = 2.9;
   - long m = 2, p = 4;
   - int 2k;
   - double x = 2 * m;
   - float y = y * 2;
   - unsigned double z = 0.0;
   - double d = 0.67F;
   - float f = 0.52L;
   - signed char = -1786;
   - char c = '$' + 2;
   - sign char h = '\111';
   - char *name = "Peter Pan";
   - unsigned char *num = "276811";

3. Cilët nga këtë emra të përmendur janë të sakta?

   - numri1
   - pese_5

- _vlera_
- njeri-tjetri
- realizimi$pagesa
- dollar_pagesa
- programimi.neobjekte
- 50me50
- deafult
- int#numer

4. Ç'far do të nxjerr në ekran kodi në vazhdim?

```
int a = 10;
cout << "\nNumri:\t" << a;
```

5. Ku është dallimi në mes të këtyre dy rreshtave në vazhdim?

```
int ch = 'A';
int ch1 = "A";
```

6. Të bëhet program që do të zgjidh këtë shprehje matematikore $(a + b)^2$. Vlerat e variablës a dhe b të futen nga tastiera, dhe rezultati të shtypet në ekran (të përdoren shprehjet **\n** dhe **\t**).

7. Cila është dalja e këtij kodi:

```
numri1 = 10;
numri2 = numri1 + 5;
numri3 = numri2 + numri1;
```

a) 50
b) 14
c) 0
d) 25

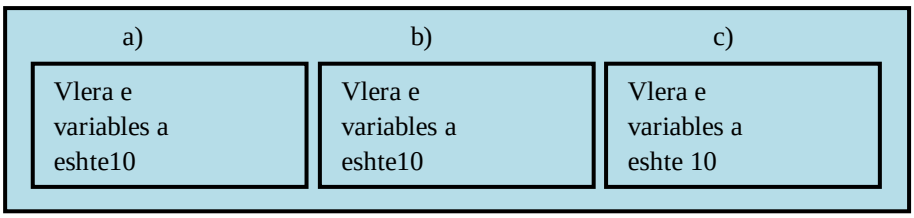
8. Të bëhet program që do të konverton njësinë matëse milimetër në centimetër dhe decimetër?

9. Cila është dalja e këtij kodi?

10.

```
#include <iostream>
```

```
using namespace std;
int main ()
{
   int a,b;
   b = 5;
   a = 2*b;
   cout << "Vlera\ne\nvariables\ta\neshte"
        << a;
   return 0;
  }
```



11. Të bëhet një program ku do të deklarohen 5 variabla të tipave të ndryshme ku të njëjtat do të shtypen në ekran në këtë mënyrë:

Variabla1
Tipi: int
Vlera: 5
// resht i zbrazët
Variabla2
Tipi: double
Vlera: 1.9 ...

Kapitulli IV

Sh p r e h j e t

Nënkapitujt:

- ✓ **Operatorët**
- ✓ **Operatorët aritmetikë**
- ✓ **Operatori i shoqërimit**
- ✓ **Shprehjet aritmetikore**
- ✓ **Vlerat logjike**
- ✓ **Operatorët relativë**
- ✓ **Operatorët logjikë**
- ✓ **Operatorët e pavlefshëm**
- ✓ **Operatorët ritës/zvogëlues**
- ✓ **Operatorët e caktimit**
- ✓ **Operatori i kushtëzuar**
- ✓ **Operatori Sizeof**
- ✓ **Operatorët me përparësi**
- ✓ **Konvertimi i tipave**
- ✓ **Shembuj**
- ✓ **Kontrollim njohurish**

Operatorët

Gjatë të shkruarit të shprehjeve për operim me të dhëna, në gjuhën C++ shfrytëzohen operatorë të ndryshëm. Më kryesorët mes tyre janë: operatori i shoqërimit, operatorët aritmetikorë, operatorët relativë (relacionalë) dhe operatorët logjikë.

Operacionet Aritmetike

C++ ofron pesë operatorë elementar aritmetikë. Këto pesë operatorë janë ilustruar në *Tabelën2.1* në vazhdim:

Operatorët aritmetikë		
Operatori	Operacioni	Shembull
+	Mbledhja	10 + 5.5 = 15.5
-	Zbritja	20 – 17 = 3
*****	Shumëzimi	4 * 4 = 16
/	Pjesëtimi	100 / 2 = 50
%	Moduli (mbetja)	10 % 4 = 2

Tabela2.1 – Operatorët aritmetikorë

Përdorimi i katër operatorëve të parë (**+**, **-**, ***** dhe **/**) është i njëjtë me përdorimin e operatorëve përkatës aritmetikorë. Kurse operatori **%** si rezultat e jep mbetjen, përkatësisht modulin nga pjesëtimi i dy numrave të plotë. Në përjashtim të modulit (mbetjes) të gjithë operatorët tjerë mund të pranojnë kombinime të numrave të plotë dhe atyre real.

Shembull janë marrë disa shprehje matematikore:

5 + 10 = 15

por ky operator pranon edhe kombinimin numër i plotë dhe real,

5.5 + 10 = 15.5

Ngjashëm edhe për operatorët tjerë,

10 - 5.5 = 4.5
10 * 5.5 = 55
10 / 5.5 = 1.8181818

Kurse moduli është mbetja dhe kjo vlen vetëm për numrat e plotë dhe atë:

20 % 3 = 2

kjo funksionon kështu:

20 / 3 = 6.666

merret vetëm numri 6 pa shtesën pas pikës dhe shumëzohet me numrin 3, kjo na jep numrin 18, tani ndryshimi në mes 20 dhe 18 është 2, kështu që mbetja apo moduli është 2.

3 * 6 = 18 + 2 = 20

Me Rëndësi! Tek operatori i pjesëtimit në qoftë pjesëtuesi dhe i pjesëtueshmi janë të tipit int, pra numra të plotë edhe herësi (rezultati) do të jetë numër i plotë. Për shembull:

10 / 4 = 2 e jo 2.5
5 / 2 = 2 e jo 2.5

Kështu që edhe pjesëtuesi dhe i pjesëtueshmi duhet të deklaruara të tipi double ose float që rezultati të jetë i saktë.

Ose si zgjidhje alternative do të ishte kjo:

```
int çmimi, sasia;
double çmimi_per_njesi;
çmimi_per_njesi = çmimi / (double) sasia;
```

Operatori i shoqërimit

Përmes operatorit të shoqërimit = (barazimit), *variablës së shkruar në anën e majtë të operatorit i shoqërohet vlera e cila gjendet në anën e djathtë* të këtij operatori. Kështu, p.sh., përmes shprehjeve:

```
z = 5;
a = 2.384;
beta = -37.4;
```

variablave z, a dhe beta u shoqërohen vlerat e shënuara në anët e djathta të barazimeve përkatëse. Para se të shkruhen në program shprehjet e

dhëna më sipër, për variablat që figurojnë në anën e majtë të barazimit duhet patjetër të deklarohen tipet varësisht nga vlerat që u shoqërohen atyre. Vlera që i është shoqëruar një variable ngel e pandryshuar derisa nuk i shoqërohet një vlerë e re.

Shprehjet aritmetikore

Me kombinimin e variablave dhe të operatorëve aritmetikorë mund të shkruhen shprehje të ndryshme aritmetikore, ngjashëm si edhe në matematikë.

Kështu, p.sh., shprehja matematikore:

x+5y-4az

në gjuhën C++ shkruhet:

6*x+3*y-3*a*z

Brenda shprehjeve aritmetikore të shkruara në gjuhën C++, sipas nevojës mund të përdoren kllapat e vogla, plotësisht njëlloj si edhe në matematikë. P.sh., shprehja matematikore:

3(x-1) - 2(y+2a+3b) – x / (y + 3)

në gjuhën C++ shkruhet:

3*(x - 1) – 2 * (y + 2 * a + 3 * b) – x / (y + 3)

Përdorimi i kllapave është i domosdoshëm veçanërisht gjatë pjesëtimit, për ta përcaktuar plotësisht të pjesëtuarin dhe pjesëtuesin, sepse shumë lehtë mund të ndodhin gabime. Kllapat duhet të përdoren edhe në rastet kur paraqiten dy operatorë aritmetikorë njëri pas tjetrit. P.sh., nëse duam ta shumëzojmë me numrin 5 me vlerën negative të variablës x, shprehja përkatëse duhet të shkruhet:

5 * (- x)

ku, duke i shfrytëzuar kllapat eliminohet përdorimi i operatorit për shumëzim dhe atij për zbritje njëri pas tjetrit.

Duke e shfrytëzuar operatorin e shoqërimit (barazimin), vlerat që fitohen nga shprehjet aritmetikore mund t'u ndahen variablave të cilat shënohen para operatorit të barazimit. Kështu, p.sh., me shprehjen e shkruar në gjuhën C++:

y=2*a+3*b-c

variablës y i shoqërohet vlera e cila llogaritet përmes shprehjes, e cila është shënuar në anën e djathtë të barazimit.

Gjatë shkruarjes së programeve në gjuhën C++, si edhe në gjuhët e tjera programuese përdoren edhe shprehje të cilat në matematikë janë të palogjikshme, siç është, p.sh., shprehja:

i = i + 1;

me të cilën në gjuhën C++ nënkuptohet rritja për 1 e vlerës së variablës **i**.

Kompjuteri, shprehjet e tilla i kupton drejt, sepse përmes tyre urdhërohet që *vlera numerike e shprehjes në anën e djathtë të barazimit t'i ndahet variablës, e cila figuron në anën e majtë të barazimit.*

Vlerat logjike

E dimë se kemi dy vlera logjike dhe ate ***true*** dhe ***flase.*** Për deklarimin e variablave të tilla na duhet tipi i të dhënave i njohur si **bool**. Pra një variabël që do të merr njërën nga dy vlerat logjike do të deklarohet kështu:

```
bool vlera;          //deklarimi
vlera = false;       //inivializimi me vleren false
```

ku **vlera** është emri i variablës apo konstantes.

Operatorët relativë

Në grupin e operatorëve relativë bëjnë pjesë operatorët të cilët përdoren për krahasimin e të dhënave, përkatësisht për testimin e raporteve mes tyre. Pas krahasimit përmes operatorëve relativë, si rezultat fitohen vlerat logjike true ose false. Nëse një relacion është i vërtetë, rezultati i tij është true, kurse për relacionet e pavërteta rezultati është false.

Operatorët relativë, të cilët përdoren në gjuhën C++, janë dhënë në *Tabelën2.1*, ku për variablat a dhe b janë marrë vlerat 4 dhe 7.

Operatori	Domethënia	Shembull	Rezultati
<	Më i vogël se	(a+1) < b	**true**
<=	Më i vogël se, ose barazi me	(3*a-2) <= (a+2*b)	**false**
==	Barazi me	(a+3)==7	**true**
>	Më i madh se	(b+2*a)>(3*a)	**true**
>=	Më i madh se, ose barazi me	(a+3*b-1) >= (4*b)	**false**
!=	Jo barazi me	(8*a-2*b) != (3*a+2)	**true**

Tabela2.2 – Operatorët relacionalë

Operatorët logjikë

Për krahasimin e më shumë shprehjeve njëkohësisht përdoren operatorët logjikë të dhënë në *Tabelën2.3*.

Operatori	Operacioni	Shembull	Rezultati
&&	Konjuksioni, AND	(x < 7) && (y ==5)	**true**
||	Disjunksioni, OR	(x != 2) || (x > 3)	**false**
!	Negacioni, NOT	!(y > 4)	**false**

Tabela2.3 - Operatorët logjikë

Këtu, rezultatet në kolonën e fundit të tabelës fitohen nëse, p.sh., për variablat x dhe y merren vlerat 2 dhe 5.

Operatori &&

Për paraqitje të operacionit logjik **AND** (DHE) shfrytëzohet operatori **&&**. Rezultati i kryerjes së operacionit **AND** mbi dy operandë do të jetë *true*, nëse vlerat e të dy operandëve janë *true*.

Operatori ||

Për paraqitje të operacionit logjik **OR** (OSE) shfrytëzohet operatori ||. Rezultati i kryerjes së operacionit logjik **OR** mbi dy operantë do të jetë *true*, nëse së paku vlera e njërit operand është *true*.

Operatori !

Për paraqitje të operacionit logjik **NOT** (JO) shfrytëzohet operatori !. Rezultati i kryerjes së operacionit logjik JO mbi një operand do të jetë *true*, nëse vlera e operandit është *false*, ose zero.

Operatorët e pavlefshëm

C++ ofron gjashtë operatorë të pavlefshëm për manipulim me bit individual në madhësi të një numri të plotë. Këto janë përmbledhur në *Tabelën 2.4*

Operatorët e pavlefshëm			
Operatori	Emri	Shembull	
~	Negacion i pavlefshëm	~'\011'	// gives '\366'
&	DHE i pavlefshëm	'\011' & '\027'	// gives '\001'
|	OSE i pavlefshëm	'\011' | '\027'	// gives '\037'
^	Ekskluziv OSE i pavlefshëm	'\011' ^ '\027'	// gives '\036'
<<	Shift i majtë i pavlefshëm	'\011' << 2	// gives '\044'
>>	Shift i djathtë i pavlefshëm	'\011' >> 2	// gives '\002'

Tabela 2.4 – Operatorët e pavlefshëm

Operatorët e pavlefshëm presin operandët e tyre të jenë madhësi integer (numër i plotë) dhe i trajtojnë ato si sekuenca bit.

Operatorët ritës/zvogëlues

Operatorët automatik ritës dhe zvogëlues janë shumë të rëndësishëm për rritjen apo zvogëlimin për 1 të një numri. Këto operatorë janë të ilustruar në *Tabelën 2.5.* Shembulli prezanton definimin e kësaj variable:

int i = 5;

Operatorët ritës dhe zvogëlues			
Operatori	Emri	Shembull	
++	Rritje automatike (parashtesë)	**++i**	*// jep 6*
++	Rritje automatike (prapashtesë)	**i++**	*// jep 5 pastaj ritet*
--	Zvogëlim automatik (parashtesë)	**--i**	*// jep 4*
--	Zvogëlim automatik (prapashtesë)	**i--**	*//jep 4 pastaj zvogëlohet*

Tabela 2.5 – operatorët ritës dhe zvogëlues

Të dy operatorët mund të përdoren në formë parashtese dhe prapashtese. Dallimi është i rëndësishëm. Kur përdoret në formë parashtese, operatori i parë është aplikuar dhe rezultati pastaj është përdorur në shprehje. Kur përdoret në formë prapashtese, së pari vlerësohet shprehja pastaj aplikohet operatori. Të dy operatorët më mirë përdoren tek numrat e plotë se sa teka ato real, edhe pse në praktikë tek numrat real këto operatorë janë shumë pak të nevojshëm.

Operatorët e caktimit

Operatori i caktimit përdoret për ruajtjen e vlerës në disa lokacione të memories (zakonisht shënohet me një ndryshore). Madhësia e majtë duhet të jetë një vlerë (lvalue), dhe madhësia (operandi) i djathtë mund të jetë një shprehje arbitrare. Ky i fundit është vlerësuar dhe rezultati është ruajtur në lokacionin e vlerës së majtë (lvalue – left value). Një lvalue (kuptim për left – value) nuk është gjë tjetër pos një hapësirë e memories ku një vlerë mund të ruhet. Lloji i vetëm i lvalue që kemi parë deri tani në këtë libër është variabël. Llojet tjera të lvalues janë përshkruar në kapitujt e ardhshëm tek printerët dhe referencat. Operatori i caktimit ka një numër variantesh, të fituar nga kombinimi i tij me aritmetikë dhe operatorët e pavlefshëm. Ja në *Tabelën 2.6* është bërë përshkrimi i tyre. Shembulli i marrë supozon se *n* është një variabël e tipi integer (numër të plotë).

Operatorët e caktimit		
Operatori	Emri	Ekuivalente me
=	n = 10	
+=	n += 10	n = n + 10
- =	n -= 10	n = n – 10
*=	n *=10	n = n * 10
/=	n /= 10	n = n / 10
%=	n %=10	n = n % 10
&=	n &= 0xF2F2	n = n & 0xF2F2
|=	n |= 0xF2F2	n = n | 0xF2F2
^=	n ^= 0xF2F2	n = n ^ 0xF2F2
<<n	n<<= 4	n = n <<= 4
>>n	n >>= 4	n = n >>=4

Tabela 2.6 – Operatorët e caktimit në C++

Një operacion i caktimit vetvetiu është një shprehje vlera e të cilës është vlera që ruhet në operuesin e saj të majtë. Pra për këtë arsye një operacion i caktimit mund të përdoret si operues i djathtë i një operacioni tjetër të caktimit. Çdo numër i caktuar mund të lidhet në këtë mënyrë për të formuar një shprehje. Për shembull:

```
int m, n, p;
m = n = p = 100;              // means: n = (m = (p = 100));
m = (n = p = 100) + 2;        // means: m = (n = (p = 100)) + 2;
```

kjo është njëlloj e zbatueshme për format e tjera të caktimit. Për shembull:

```
m = 100;
m += n = p = 10;  // do te thotë: m = m + (n = p = 10);
```

Operatori i kushtëzuar ?

Në gjuhën C++ përdoret një operator i veçantë dypjesësh **? :**, për llogaritje të kushtëzuar. Shprehjet që formohen duke e shfrytëzuar operatorin e kushtëzuar në formë të përgjithshme duken kështu:

```
y = k ? a : b;
```

Kompjuteri, sa herë që i takon shprehjet e kësaj forme, nëse kushti **k** është i saktë, variablës **y** ia ndan vlerën e shprehjes **a**, përndryshe ia ndan vlerën e shprehjes **b**, gjë që në matematik shprehet kështu:

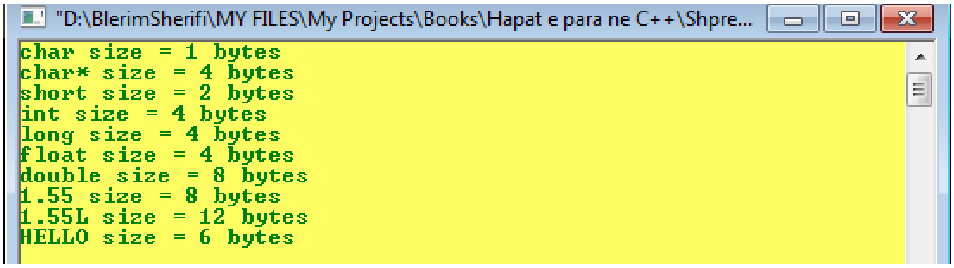
Operatori Sizeof

C++ ofron një operatorë të dobishëm, sizeof, për llogaritje të madhësisë së çdo artikulli të të dhënave ose tipave. Ajo merr një operand të vetëm i cili mund të jetë emër i tipit (p.sh. int) ose një shprehje (p.sh. 100) dhe kthen madhësinë e subjektit të specifikuar në bajt. Rezultati është i varur krejtësisht nga makina. Shembulli në vazhdim ka të ilustruar përdorimin e operatorit sizeof.

Zgjidhje:

```
#include <iostream.h>
using namespace std;
int main (void)
{
        cout << "char size = " << sizeof(char) << " bytes\n";
        cout << "char* size = " << sizeof(char*) << " bytes\n";
        cout << "short size = " << sizeof(short) << " bytes\n";
        cout << "int size = " << sizeof(int) << " bytes\n";
        cout << "long size = " << sizeof(long) << " bytes\n";
        cout << "float size = " << sizeof(float) << " bytes\n";
        cout << "double size = " << sizeof(double) << " bytes\n";
        cout << "1.55 size = " << sizeof(1.55) << " bytes\n";
        cout << "1.55L size = " << sizeof(1.55L) << " bytes\n";
        cout << "HELLO size = " << sizeof("HELLO") << " bytes\n";
}
```

Pasi të ekzekutojmë kodin në ekran do të kemi këtë rezultat, pra madhësinë në bajt të tipave apo shprehjeve të përdorura në shembull:



Operatorët me përparësi

Renditja në të cilën operatorët janë përcaktuar në një shprehje është e rëndësishme dhe e përcaktuar nga rregullat e përparësisë. Këto rregulla i ndajnë C++ operatorët në numrin e niveleve të përparësisë (prioritetit). Operatorët në nivelin e lartë kanë përparësi ndaj operatorëve të nivelit të ulët.

Nivelet e operatorëve me përparësi			
Niveli	**Operatori**	**Lloji**	**Renditja**
I lartë	: :	Unar	Bashkë
	( ) [ ] -> .	Binar	Nga e majta në të djathtë
	+ - ++ -- ! ~ * & new delete Sizeof( )	Unar	Nga e djathta në të majtë
	->* .*	Binar	Nga e majta në të djathtë
	* / %	Binar	Nga e majta në të djathtë
	+ -	Binar	Nga e majta në të djathtë
	<< >>	Binar	Nga e majta në të djathtë
	< <= > >=	Binar	Nga e majta në të djathtë
	== !=	Binar	Nga e majta në të djathtë
	&	Binar	Nga e majta në të djathtë
	^	Binar	Nga e majta në të djathtë
	|	Binar	Nga e majta në të djathtë
	&&	Binar	Nga e majta në të djathtë
	||	Binar	Nga e majta në të djathtë
	? :	Trinar	Nga e majta në të djathtë
	= += *= ^= &= <<=	Binar	Nga e djathta në të majtë

		-=	/=	%=	|=	>>=		
I ulët	,						Binar	Nga e majta në të djathtë

Tabela 2.7 – Nivelet e operatorëve me prioritet

Për shembull, në

a == b + c * d

***** (prodhimi) do të vlerësohet në fillim, pasi ***** (prodhimi) ka nivel më të lartë të prioritetit se sa + dhe ==. Pasi të kryhet prodhimi rezultati i ndahet b pasi + ka prioritet më të lartë se ==, dhe në fund vlerësohet ==. Rregullat e prioritetit (përparësisë) mund të merren duke përdorur edhe kllapat. Për shembull:

a == (b + c) * d

kjo bën që + të vlerësohet para *****.

Operatorët me të njëjtin nivel të përparësisë vlerësohen në bazë të renditjes që është dhënë në kolonën e fundit të *Tabelës 2.9*. Për shembull:

a = b += c

rendi i vlerësimit në këtë rast është nga e djathta në të majtë, kështu që në fillim vlerësohet b += c, ndjekur nga a = b.

Konvertimi i tipave

Një vlerë në ndonjë nga tipat e përmendur deri tani mund të konvertohen (type – cast) në ndonjërin nga tipat tjerë. Për shembull:

```
(int) 3.14                  // konverton 3.14 në int dhe jep 3
(long) 3.14                 // konverton 3.14 në long dhe jep 3L
(double) 2                  // konverton 2 në double dhe jep 2.0
(char) 122      // konverton 122 në char dhe jep karakterin e 122
(unsigned short) 3.14       // jep 3 si një unsigned short
```

Siç shihet nga këto shembuj, identifikatorët e tipave të përmendur deri tani mund të përdoren si operatorë të tipit. Operatorët e tipit janë unarë (p.sh., marrin një madhësi) dhe shfaqen brenda kllapave në të majtë

të madhësisë së tyre. Kjo quhet konvertim i qartë i tipit. Kur emri i tipit është vetëm një fjalë, një shënim alternativ mund të përdoret që kllapat të jenë përreth madhësisë (operandit):

```
int (3.14)          // same as: (int) 3.14
```

Në disa raste, C++ gjithashtu kryen implicit type conversion (nënkupton konvertimin e tipit). Kjo ndodh kur vlerat e tipave të ndryshëm janë të përzier në një shprehje.

Për shembull:

```
double d = 1;       // d receives 1.0
int i = 10.5;       // i receives 10
i = i + d;          // means: i = int(double(i) + d)
```

në shembullin e fundit **i + d** përfshin tipa që nuk përputhen, kështu që më parë **i** është konvertuar në **double** dhe më pas i shtohet **d**. Rezultati është **double** që nuk përputhet me **i** në anën e majtë të shprehjes, pra ajo konvertohet në **int** para se ti ndahet **i**.

Rregullat e mësipërme përfaqësojnë disa raste të thjeshta, por të zakonshme për konvertim të tipit. Raste më komplekse do të shqyrtohen më vonë, kur ne të kemi folur për tipat e tjera të të dhënave dhe klasat.

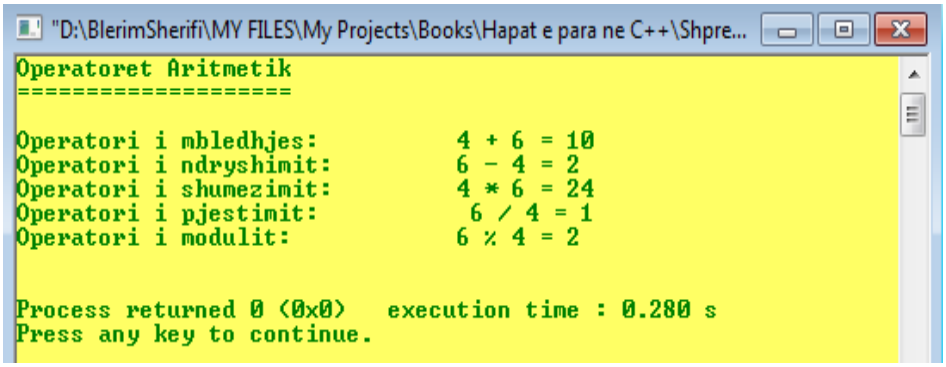
Shembuj

Shembull1: Të bëhet program ku do të ilustrohen të gjithë operatorët aritmetik me një shembull, dhe rezultatet të nxirren në ekran.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   int mbledhja, zbritja, moduli;
   double pjestimi, shumezimi;

   int a, b;
   a = 4, b = 6;

   mbledhja = 4 + 6;              // operatori i mbledhjes
   zbritja = 6 - 4;               // operatori i ndryshimit
   shumezimi = 4 * 6;             // operatori i shumëzimit
   pjestimi = 6 / 4;              // operatori i pjesëtimit
   moduli = 6 % 4;                // operatori i modulit (mbetjes)

   cout << "Operatoret Aritmetik\n"
        << "====================\n"
        << "\nOperatori i mbledhjes:  \t"
        << a << " + " << b << " = " << mbledhja
        << "\nOperatori i ndryshimit: \t"
        << b << " - " << a << " = " << zbritja
        << "\nOperatori i shumezimit: \t"
        << a << " * " << b << " = " << shumezimi
        << "\nOperatori i pjestimit:  \t "
        << b << " / " << a << " = " << pjestimi
        << "\nOperatori i modulit:    \t"
        << b << " % " << a << " = " << moduli
        << "\n\n";
   return 0;
}
```

Rezultati pas ekzekutimit do të duket kështu:

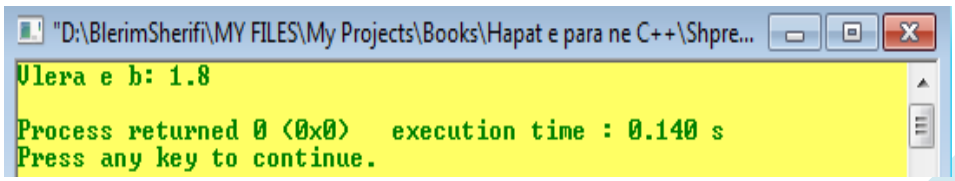


Shembull2: Të bëhet program që do të llogarit këtë shprehje **a/b**, ku **a** dhe **b** janë numra të plotë dhe vlera e tyre është : **a = 9, b = 5;** të përdoret konvertimi i tipave.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   int a, b;
   double c;

   a = 9;
   b = 5;
   c =  (double) a / b;                  // Konvertimi i shprehjes a / b nga int në double
   cout << "Vlera e b: " << c << endl;
   return 0;
}
```

Rezultati në ekran pas ekzekutimit do të duket:



Shembull3: Të bëhet program për llogaritjen e kësaj shprehje matematikore:

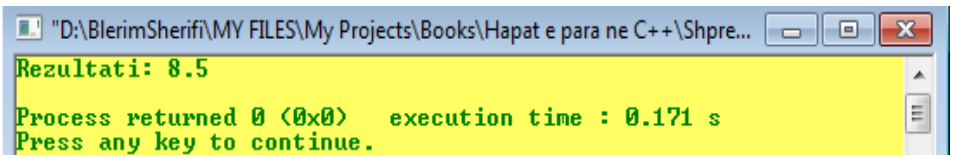
$$4a + (100 / b) - (b + c * 3)$$

ku **a = 10, b = 20, c = 5.5** rezultati të nxirret në ekran.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   int a = 10, b = 20;
   double c = 5.5;
   double rez;
   rez = 4 * a + ( 100 / b ) - ( b + c * 3);          // llogaritja e formulës
   cout << "Rezultati: " << rez << endl;
   return 0;
}
```

Rezultati në ekran do të duket kështu:



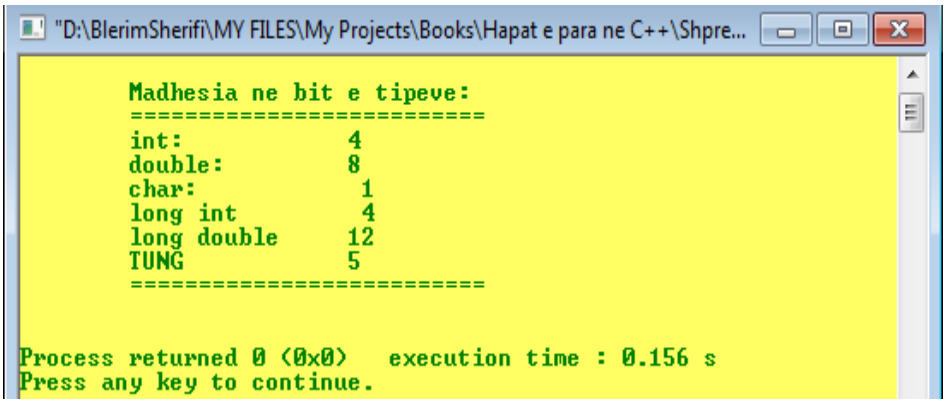
Shembull4: Të bëhet program ku në ekran do të nxjerr madhësinë në bit të tipeve: **int, double, char, long int, long double, TUNG.**

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   cout << "\n\tMadhesia ne bit e tipeve: "
        << "\n\t=========================";
   cout << "\n\tint:       \t"   << sizeof(int)
          << "\n\tdouble:    \t"   << sizeof(double)
```

```
      << "\n\tchar:     \t "  << sizeof (char)
      << "\n\tlong int   \t "  << sizeof(long int)
      << "\n\tlong double\t"   << sizeof(long double)
      << "\n\tTUNG       \t"   << sizeof("TUNG")
      << "\n\t==========================="
      << "\n\n";
   return 0;
}
```

Pas ekzekutimit rezultati në ekran do të duket kështu:



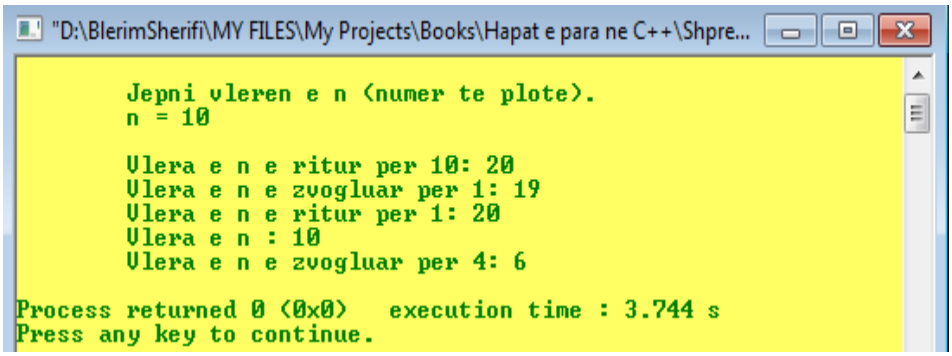
Shembull5: Të bëhet program ku do të deklarohet një variabël me emrin ***n*** dhe vlera e kësaj variable të futet nga tastiera, kurse tipi i saj të jetë **int**. Pastaj kjo variabël të ritet për **10**, dhe të nxirret në ekran. Pastaj të zvogëlohet vlera e saj për **1** dhe të nxirret në ekran përsëri. Përsëri, të ritet vlera e saj për **1** dhe të nxirret në ekran. Pastaj të pjesëtohet me **2** dhe të nxirret në ekran, dhe në fund ti hiqet vlera **4** dhe të nxirret në ekran. *Vërejtje: të përdoren operatorët* ***ritës/zvogëlues.***

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   int n;
   cout << "\n\tJepni vleren e n (numer te plote)."
```

```
        << "\n\tn = ";
    cin >> n;

    n = n + 10;                                          // rritja e n për 10
    cout << "\n\tVlera e n e ritur per 10: " << n;
    --n;                                                 // zvogëlimi i n për 1
    cout << "\n\tVlera e n e zvogluar per 1: " << n;
    ++n;                                                 // rritja e n për 1
    cout << "\n\tVlera e n e ritur per 1: " << n;
    n = n / 2;                                           // pjesëtimi i n me 2
    cout << "\n\tVlera e n : " << n;
    n = n - 4;                                           // zvogëlimi i n për 4
    cout << "\n\tVlera e n e zvogluar per 4: " << n << endl;
    return 0;
}
```

Dalja në ekran do të duket kështu:



Kontrollim njohurish

1. Cilët operatorë më së shpeshti përdoren në C++?

2. Cila është dalja e këtij kodi:

```
a = 10 + 5;
a = a – 5;
--a;
cout << a;
```

a) 9 b) 7 c) 12 d) 0

3. Të bëhet program që do të llogaris këtë formulë matematikore:

10a / (5b – 10) * (2 + c)

ku **a = 10, b = 3, c = 3.**

4. Cila do të jetë dalja e këtij kodi:

```
#include <iostream>
using namespace std;
int main ()
{
   int n,numri1;
   numri1 = 5;
   n = numri1;
   cout << "numri1: " << n;
   --numri1;
   cout << numri1;
   numri1 = numri1 + n;
return 0;
}
```

a) 45 **b)** 53 **c)** 52 **d)** 54

5. Të bëhet program ku do të nxjerr se sa bajt në memorie zënë këto tipa të të dhënave: **shor int, short double, float, char*, PROGRAMIM**.

6. Cila është dalja e këtij kodi:

```
#include <iostream.h>
using namespace std;
int main (void)
{
cout << sizeof(char) << “ ";
cout << sizeof(short) << “ ";
cout << sizeof(long) << " ";
cout << sizeof(1.55) << " ";
cout << sizeof(1.55L);
}
```

a)1 1 4 3 12 b) 2 3 4 8 10 c) 5 0 2 1 11 d) 1 2 4 8 12

Kapitulli V

D e k l a r a t a t

Nëntitujt:

- ✓ **Çka janë deklaratat**
- ✓ **Deklaratat *e shprehjes***
- ✓ **Deklaratat *null***
- ✓ **Deklaratat *e përbëra***
- ✓ **Deklaratat *e përzgjedhjes***
- ✓ **Deklaratat *e përsëritjes***
- ✓ **Deklarata *kapërcyeke***
- ✓ **Deklarata *deklaruese***

Çka janë Deklaratat

C++ deklaratat janë elemente të programit që kontrollojnë se si dhe në ç'far rendi janë manipuluar objektet. Kemi këto lloje të deklaratave:

- **Deklaratat e shprehjes**. Këto deklarata vlerësojnë një shprehje për efektet e saj anësore ose për vlerën e saj të kthimit. Për shembull: 2x*3y^2;
- **Deklaratat null.** Këto deklarata mund të sigurohen kur deklarata kërkohet nga sintaksa e C++, por ku nuk ka që të ndërmerret ndonjë aksion.
- **Deklaratat e përbëra.** Këto deklarata janë grupe të deklaratave të mbyllura me kllapa gjarpëruese ({}).
- **Deklaratat e përzgjedhjes.** Këto deklarata kryejnë një test; ata ekzekutojnë një pjesë të kodit nëse testi vlerëson *true* (saktë, jo-zero). Dhe mund të ekzekutojnë një pjesë tjetër të kodit nëse testi vlerëson *false* (jo-saktë ose zero).
- **Deklaratat e përsëritjes.** Këto deklarata sigurojnë një ekzekutim të përsëritur të një blloku të kodit përderisa një kriter specifik është plotësuar.
- **Deklaratat kërcyese (kapërcyeke).** Këto deklarata për më tepër transferojnë kontrollin në një vend tjetër në funksion ose kthejnë kontrollin prej funksionit.
- **Deklaratat deklaruese.** Deklarimet paraqesin një emër në një program (p.sh., emërtimi i ndryshoreve).

Deklaratat e shprehjes

Deklaratat e shprehjes shkaktojnë shprehje që të vlerësohet. Nuk ka transfero të kontrollit ose ndonjë përsëritje si rezultat i një shprehje.

Sintaksa e deklaratave të shprehjes është e thjeshtë:

[shprehja] ;

Të gjitha shprehjet në një deklaratë të shprehjes vlerësohen dhe të gjitha efektet anësore kompletohen përpara se deklarata tjetër të ekzekutohet.

Deklaratat më të zakonshme janë shprehje caktimi dhe thirrje funksioni. Po ashtu kemi e dhe deklaratë e shprehjes boshe, e cila shënohet vetëm me pikëpresje (;), që gjithashtu i referohet deklaratës *null*.

Shembull:

```
int B=10;
double A;
A += B;          // deklaratë e shprehjes
```

Null deklaratat (boshe)

Deklarata **null** është një deklaratë shprehje por me mungesë të shprehjes. Kjo është e rëndësishme kur sintaksa e gjuhës bën thirrje për një deklaratë por nuk ka shprehje që duhet për të vlerësuar. Kjo përbëhet vetëm na një pikëpresje '**;**'.

Deklaratat **null** janë përdorur zakonisht si mbajtës të vendit (ang. placeholders) në deklaratat përsëritëse ose si deklarata ku do të vendosim etiketa (ang. labels) në fund të deklaratave të komplekse ose funksioneve.

Shembulli në vazhdim inicializon elementet e një vektori çmimesh. Nga se inicializimet ndodhin pa përdorimin e *shprehjeve* të deklaratës ***for***, është e nevojshme një deklaratë për të përfunduar sintaksën e deklaratës ***for***, e kjo deklaratë është **null** deklarat '**;**':

```
for (i = 0; i < 3; price[i++] = 0)
        ;          //deklarata null për të përfunduar for.
```

Pra, këtu nuk nevojitet as një operacion.

Një deklaratë **null** mund të përdoret kur është i nevojshëm një emërtim (ang. label) para përfundimit të bllokut të deklaratës. Për shembull:

```
void func(void)
{
      if (error_detected)
          goto depart;
      depart: ;                    // deklarata null nevojitet
}
```

Deklaratat e përbëra

Deklarata e përbërë përbëhet na asnjë ose shumë C++ deklarata të mbyllur nga kllapa gjarpëruese "**{}**". Kjo është një koncept thelbësor në C++ dhe është qendrore për idenë e ndërtimit të folesë.

Për shembull këto deklarata janë pjesë e deklaratave të tjera siç është *if* deklarata:

```
if ( a > b )
{                          // fillimi i deklaratës së përbërë
      Float_t temp = b;
      b = a;
      a = temp;
}                          // mbarimi i deklaratës së përbërë
```

Deklaratat e përzgjedhjes

Deklaratat e përzgjedhjes përdoren për të përzgjedhur mes deklaratave alternative (ose blloqeve të deklaratave) që janë ekzekutuar me kusht. Ka tri deklarata të përzgjedhjes në C++ dhe ato janë:

- Deklarata **if** (dhe disa variacione të sajë),
- Deklarata **switch** dhe
- Operatori **ternar** (i përdorur rrallë).

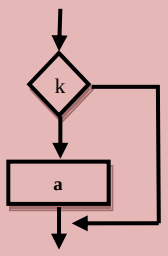
Deklarata if

Deklarata themelore përmes së cilës realizohen degëzimet e ndryshme në programe është deklarata **if**. Gjatë shkrimit të kësaj deklarate mund të paraqiten disa raste të degëzimeve: *të zakonshme, të përbëra, të shumëfishta* dhe *të ndërthurura*.

Më poshtë është paraqitur një skemë ku në të kemi të katër llojet e paraqitjes së deklaratës **if** në formën e tyre të përgjithshme.

Degëzim i zakonshëm:

```
if (k)
a;
```



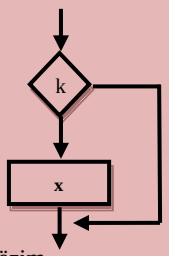
ku janë:

k - kushti për degëzim.

a - komandë e cila ekzekutohet, nëse plotësohet kushti k.

Degëzim i përbërë:

```
if (k)
{
x;
}
```



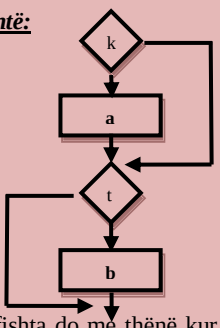
ku janë:

k - kushti për degëzim.

x - komanda e përbërë, e cila ekzekutohet, nëse plotësohet kushti k. Kurse kllapa gjarpëruese '**{**' tregon fillimin e kushtit dhe '**}**' mbarimin e kushtit

Degëzim i shumëfishtë:

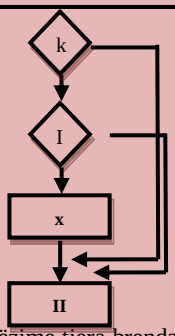
```
if (k)
      a;
if (t)
      b;
```



Degëzime të shumëfishta do më thënë kur në program kemi më shumë se një degëzim.

Degëzim i ndërthurur:

```
if (k)
      if(I)
            x;
      else
      {
            II;
      }
```



ku:

I dhe **II** janë degëzime tjera brenda degës së parë dhe të dytë.

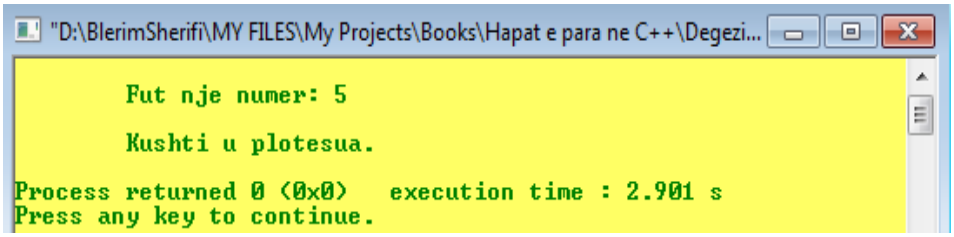
Degëzime të zakonshme

Nëse në degët e deklaratës **if** ka vetëm nga një komandë thuhet se deklarata **if** është e zakonshme. Në vazhdim kemi marrë disa shembuj për të kuptuar më mirë përdorimin e kësaj deklarate në program.

Shembull: Të bëhet program ku do të futet një numër nga tastiera dhe nëse numri është më i madh se 0 në ekran të nxirret mesazhi “Kushti u plotësua.”.

```
#include <iostream>
using namespace std;
int main ()
{
   int numri1;

   cout << "\n\tFut nje numer: ";
   cin >> numri1;

   if (numri1>0)                          // kushti nëse numri1 është më i madh se 0
   cout << "\n\tKushti u plotesua.\n";     // nëse plotësohet kushti nxirret teksti në
                                             ekran
   return 0;
}
```

Rezultati në e kran do të del “Kushti u plotësua.” në qoftë se numri i futur do të jetë më i madh se 0 dhe rezultati do të duket kështu:



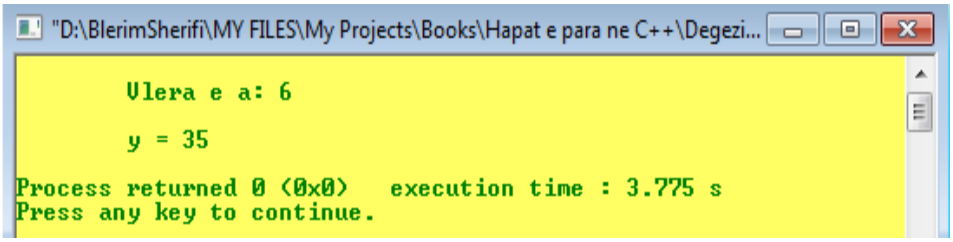
pra në qoftë se numri është më i vogël se 0 atëherë në ekran nuk do të del asgjë.

Në vazhdim është marrë një shembull tjetër ku do të llogaritet një shprehje matematikore:

Shembull: Të bëhet program për llogaritjen e kësaj shprehje y = 5a + 5, ku a > 5.

```
#include <iostream>
using namespace std;
int main ()
{
   int a;
   int y;
   cout << "\n\tVlera e a: ";
   cin >> a;

   if (a>5)                    // pra kushti nëse a është më e madhe se 5
      y = 5 * a + 5;           // nëse po atëherë kryhet llogaritja e shprehjes

   cout << "\n\ty = " << y << endl;
   return 0;
}
```

Rezultati i cili do të shfaqet në ekran po që se plotësohet kushti do të jetë:



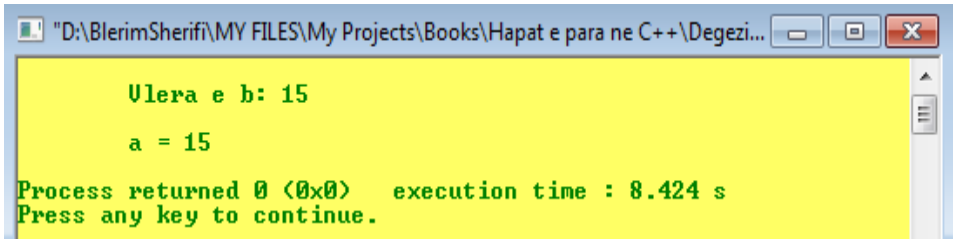
Ndërsa në qoftë se nuk plotësohet kushti programi vazhdon më poshtë ku nuk ka komanda për të ekzekutuar dhe përfundon, andaj në ekran nuk nxirret as një rezultat.

Një shembull tjetër ku do të kryhen disa komanda pas kushti:

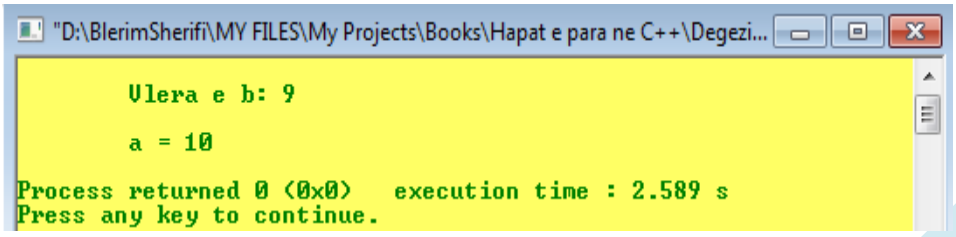
Shembull: Të bëhet program ku do të futet një vlerë nga tastiera dhe në qoftë se vlera është më e madhe se vlera e variabëlit **a**, variabëli **a** të jetë i barabartë me vlerën e futur. Në fund të nxirret në ekran variabëli **a**. Vlera fillestare e variabëlit a të jetë 10.

```
#include <iostream>
using namespace std;
int main ()
{
   int a = 10;
   int b;
   cout << "\n\tVlera e b: ";
   cin >> b;

   if (b>a)                    // kushti i cili nëse b > a
      a = b;                   // nëse plotësohet kushti atëherë a merr vlerën e b

   cout << "\n\ta = " << a << endl;          // komanda që plotësohet pas kushtit
   return 0;
}
```

Dalja në ekran po që se do të plotësohet kushti do të duket:



Por në qoftë se nuk plotësohet kushti programi do të vazhdon dhe dalja do të duket kështu:



Me komanda të përbëra

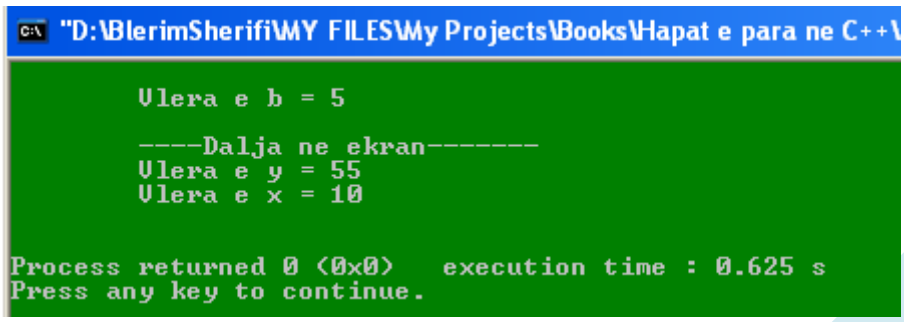
Shpesh herë nevojitet që pasi të plotësohet kushti (deklarata *if*), të ketë më shumë komanda, të cilat këtu do t'i quajmë *komanda të përbëra*. Këto komanda patjetër duhet të shënohen brenda kllapave të mëdha ({,}).

Shembull: Të bëhet program që do të llogaris vlerën e x dhe y nëqoftë se vlera hyrëse e variablës b është më e madhe se 0.

y = 10b +5 x = 5 + b

```
-
# include <iostream>
using namespace std;
int main ()
{
int b;
double y,x;
cout << "\n\tVlera e b = ";
cin >> b;
if (b > 0)                    // kushti qe duhet te plotesohet
{                             // fillimi i trupit te kushtit
y = 10 * b + 5;       // x dhe y janë komanda por llogariten si komanda te pëerë
x = 5 + b;  // pasi kryhen brenda një kushti pra kur b të jetë më e madhe se zero.
}                             // mbarimi i kushtit
cout << "\n\t----Dalja ne ekran-------";
cout << "\n\tVlera e y = " << y
<< "\n\tVlera e x = " << x
<< "\n\n";
return 0;
```

Dalja në ekran pasi të plotësohet kushti, pra b të jetë më e madhe se 0 do të jetë:



Por në qoftë se nuk plotësohet kushti atëherë programi vazhdon më poshtë ku nuk do të ketë komanda për të ekzekutuar dhe në dalje nuk do të nxirret gjë.

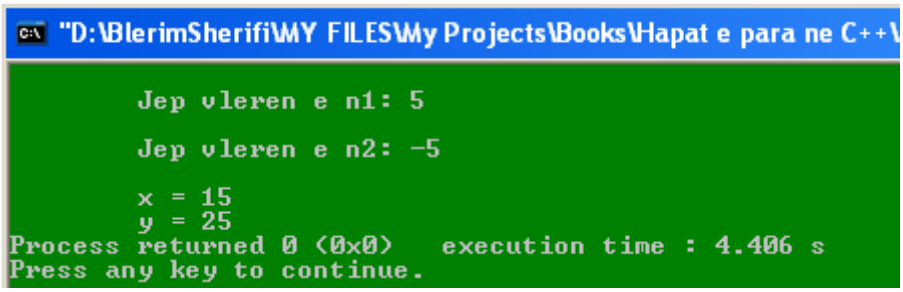
Degëzime të shumëfishta

Degëzime të shumëfishtë kemi atëherë kur në program kemi më shumë se një deklaratë **if**. Pra në program mund të kemi më shumë deklarata **if** të cilat ekzekutohen në vete. Për të kuptuar më mirë në vazhdim kemi marrë disa shembuj.

Shembull: Të bëhet program ku në të do të llogariten x dhe y. Të deklarohen dy të panjohura n1 dhe n2. Vlera e tyre të futet nga tastiera dhe nëse vlera e ndryshores n1 është më e madhe se 0 të llogaritet x = 10 + n1, dhe nëse vlera e ndryshores n2 < 0 të llogaritet y = -5 * n2. Në qoftë se kushtet nuk plotësohen vlerat fillestare për x dhe y të merren 0. Më pas x dhe y të nxirren në ekran.

```
# include <iostream>
using namespace std;
int main ()
{
   float x=0,y=0;
   short n1,n2;

   cout << "\n\tJep vleren e n1: ";
   cin >> n1;
   cout << "\n\tJep vleren e n2: ";
   cin >> n2;

   if (n1 > 0)                          // Kushti i parë
      x = 10 + n1;

   if (n2 < 0)                          // Kushti i dytë
      y = -5 * n2;

   cout << "\n\tx = " << x;
   cout << "\n\ty = " << y;

   return 0;
}
```

Pasi të ekzekutohet programi në qoftë se vlerat hyrëse për **n1** dhe **n2** i japim **5** dhe **-5**, atëherë rezultati në ekran do të duket kështu:



Por nëse kushtet nuk plotësohen atëherë vlera e **x** dhe **y** do të jetë 0.

Degëzime të ndërthurura

Degëzime të ndërthurura janë ato degëzime ku deklarata **if** brenda vetes nuk përmban komanda të thjeshta por përmban një komandë tjetër **if**. Pra, kemi kusht brenda kushtit. Kjo do të thotë që nëse plotësohet kushti i parë pastaj përsëri duhet të plotësohet edhe kushti tjetër dhe më pas të ekzekutohet ndonjë komandë.

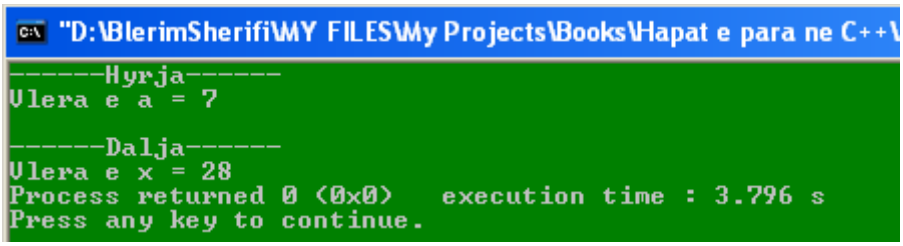
Në vazhdim do të marrim një shembull ku më mirë do të ilustrojmë deklaratën **if** me degëzim të ndërthurur.

Shembull: Të bëhet program që do të llogaris shprehjen **x = 4 * a**, në qoftë se më parë vlera e **a > 0** dhe më pas vlera e **a < 10**. Rezultati të nxirret në ekran.

```
# include <iostream>
using namespace std;

int main ()
{
   int a,x;
   cout << "Vlera e a = ";
   cin >> a;
   if (a >0)                    // me të hijezime është paraqitur një degëzim i ndërthurur
      if (a<10)                 // pra pasi të kryhet kushti i parë shqyrtohet kushti i dytë.
      {
```

```
        x = 4 * a;
        cout << "Vlera e x = " << x;
    }
  return 0;
}
```

Pas ekzekutimit të programit dalja në ekran nëse plotësohen të dy kushtet do të jetë:



Ndërsa në qoftë se plotësohet kushti i parë por jo edhe kushti i dytë atëherë kjo do të ishte e njëjtë sikurse nuk plotësohet kushti i parë dhe në ekran nuk do të nxirret asgjë.

Deklarata swich

Deklarata **switch** është e njëjtë si deklarata **if** por ajo që e bën këtë deklaratë të dalloj është mundësia për të zgjedhur ndërmjet alternativave të shumta. Forma e përgjithshme e komandës **switch** është:

```
switch (shprehja)
{
        case konstant1:
                komandat;
        case konstant2
                komandat;
        …
        case konstantn:
                komandat;
        default:
                komandat;
}
```

Në fillim ekzekutohet ***shprehja*** (e quajtur edhe si tagu i **switch**-it), dhe rezultati krahasohet me secilin konstant para fjalës kyçe **case**. Tek degëzimi **switch** kllapat gjarpëruese janë të domosdoshme.

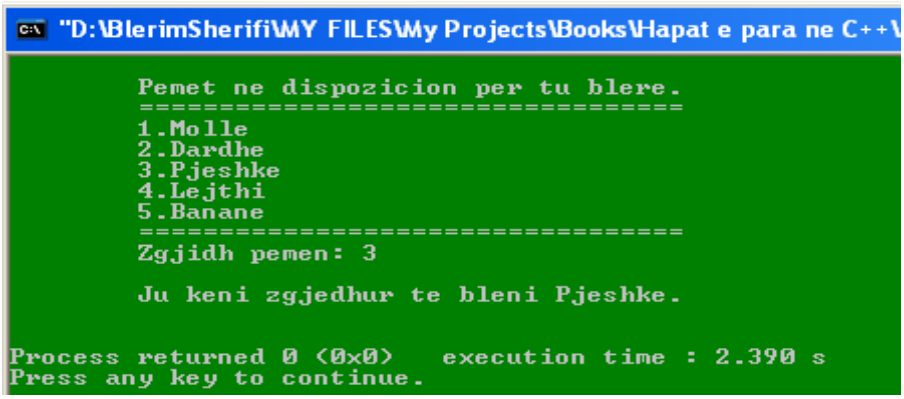
Në vazhdim do të marrim një shembull ku do të përdorim deklaratën **switch**, për të kuptuar më mirë rolin e saj në program. Si shembull do të marrim një shitore perimesh, ku përdoruesi do të mund të zgjedh një pemë dhe varësisht se cilën pemë e ka zgjedhur ajo do të shfaqet në ekran.

Shembull: Të bëhet program ku në ekran do të nxirren 5 artikuj pemësh, me numër rendor duke filluar nga numri 1. Pastaj të deklarohet një e panjohur me emrin pema e tipit short, ku përdoruesi do të fut numrin e pemës së tij dhe ajo më pas do të shfaqet në ekran.

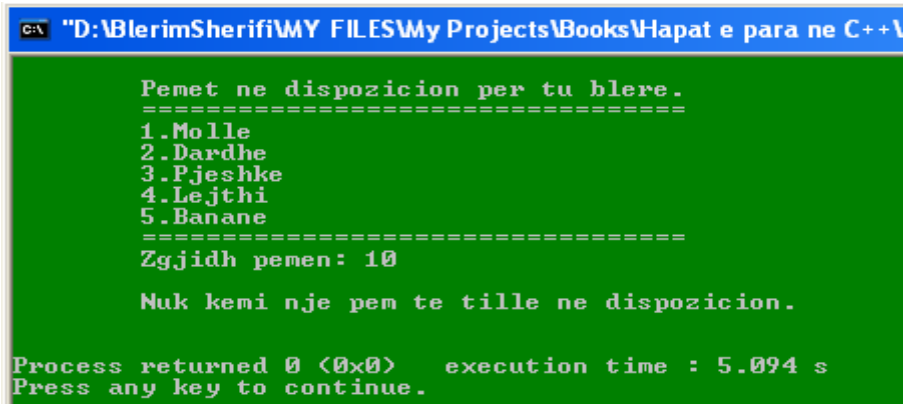
```
# include <iostream>
using namespace std;
int main ()
{
   short pema;
//--------------------Pemet-------------------------------------------------
   cout << "\n\tPemet ne dispozicion per tu blere.";
   cout << "\n\t==================================";
   cout << "\n\t1.Molle"
        << "\n\t2.Dardhe"
        << "\n\t3.Pjeshke"
        << "\n\t4.Lejthi"
        << "\n\t5.Banane";
   cout << "\n\t==================================";
//-------------------Zgjedhja e pemës ----------------------------------------
   cout << "\n\tZgjidh pemen: ";
   cin >> pema;

//-------------------Nxjerrja në ekran përmes komandës për degëzim switch----------
   switch (pema)
   {
      case 1: cout << "\n\tJu keni zgjedhur te bleni Molle.";
         break;
      case 2: cout << "\n\tJu keni zgjedhur te bleni Dardhe.";
         break;
      case 3: cout << "\n\tJu keni zgjedhur te bleni Pjeshke.";
         break;
```

```
        case 4: cout << "\n\tJu keni zgjedhur te bleni Lejdhi.";
                break;
        case 5: cout << "\n\tJu keni zgjedhur te bleni Banane.";
                break;
        default: cout << "\n\tNuk kemi nje pem te tille ne dispozicion.";
    }
    cout << "\n\n";
    return 0;
}
```

Pas ekzekutimit të programit dalja në ekran do të duket kështu:



Por nëse parashtrohet pyetja: Çfarë do të ndodh nëse fusim ndonjë numër tjetër që nuk gjendet në shitore? Për këtë gjë kujdeset pjesa e fundit e deklaratës **switch** gjegjësisht **default** që ekzekutohet në rastin kur nuk plotësohen komandat paraprake, kështu që në ekran do të nxirret teksti "*Nuk kemi një pemë të tillë në dispozicion.*". Dalja do të duket kështu:



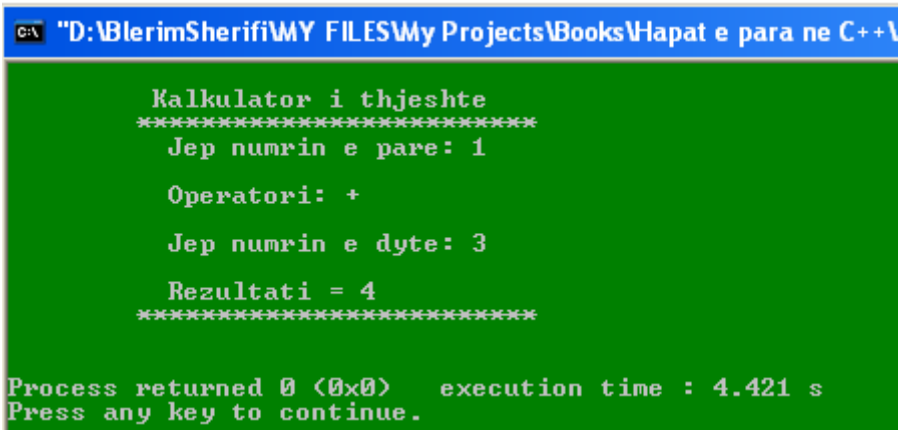
Ja në vazhdim do të krijojmë një kalkulatorë të thjeshtë që manipulon me dy numra varësisht se cilin operator kemi futur ne.

Shembull: Të bëhet program që operon me dy numra varësisht se cilin operator kemi futur nga tastiera. Të përdoret deklarata **switch** për zgjedhje në mes të alternativave.

```
# include <iostream>
using namespace std;
int main ()
{
   int a, b, c;
   char ch;
   string v = "\n\t**************************";
   cout << "\n\t Kalkulator i thjeshte";
   cout << v;
   cout << "\n\t  Jep numrin e pare: ";
   cin >> a;
   cout << "\n\t  Operatori: ";
   cin >> ch;
   cout << "\n\t  Jep numrin e dyte: ";
   cin >> b;
```

```
    switch (ch)
  {
     case '+': c = a + b;    cout << "\n\t  Rezultati = " << c;
     break;
     case '-': c = a - b;    cout << "\n\t  Rezultati = " << c;
     break;
     case '*': c = a * b;    cout << "\n\t  Rezultati = " << c;
     break;
     case '/': c = a / b;    cout << "\n\t  Rezultati = " << c;
     break;

     default: cout << "\n\t Operator i gabuar!"

              << "\n\t Ju mund te zgjidhni vetem keto operator: (+, -, *, /).";
  }
  cout << v << "\n\n"
  return 0;
}
```

Pas ekzekutimit dalja në ekran do të duket si në figurën e poshtme po që se dëshirojmë të mbledhim numrin 1 me numrin 2:



Programi njëjtë funksionon edhe për operatorët tjerë. Një vërejtje mund të jetë tek operatori i shumëzimit se në qoftë se dëshirojmë të pjesëtojmë numrin 5 me 2, atëherë rezultati do të del 2, pasi pjesa pas pikës dhjetore nuk llogaritet sepse ndryshoret i kemi deklaruar të tipit int (integer, numër i plotë).

Deklarata *ternary*

Kjo është një deklaratë tjetër rrallë e përdorur në C++. Në formën e sajë të thjeshtë, ajo është shumë e ngjashme me deklaratën *if* bazuar në kushtin *true/false* që përcakton se cili nga dy deklaratat është ekzekutuar. Një shembull tregon më poshtë:

```
(x > y) ? cout << x << " is greater than " << y << endl:
          cout << y << " is greater than or equal to " << x << endl;
```

Vini re se së pari është dhënë kushti në kllapa, dhe që pasohet nga një pikëpyetje '**?**' dhe deklarata që do të ekzekutohet nëse kushti është *true*. Kjo është e ndarë nga deklarata që do të ekzekutohet nëse kushti është *false* me anë të dy pikave '**:**'. Deklarata duhet patjetër, si gjithmonë, të ndërpritet nga një pikëpresje '**;**'.

Deklaratat përsëritëse

Deklaratat përsëritëse shkaktojnë deklarata (ose deklarata të përbëra) për tu ekzekutuar një ose më shumë herë, në varësi të disa kritereve të përfundimit të ciklit. Kur këta deklarata janë deklarata të përbëra, ata ekzekutohen në renditje, përveç rasteve kur kemi hasur deklaratën *break* ose deklaratën *continue.*

C++ ofron katër deklarata përsëritëse dhe ato janë:

- Deklarata **While**,
- Deklarata **do**,
- Deklarata **for** dhe
- Deklarata **range-based for**.

Deklarata *while*

Deklarata **while** (thirret cikli **while**), krijon një mënyrë të përsëritjes përderisa të jetë plotësuar kushti. Kjo është njëra nga ciklet e përsëritjes në C++. Forma e përgjithshme e deklaratës **while** është:

```
while (shprehja)
komanda;
```

Në fillim vlerësohet *shprehja* e parë (e quajtur kushti i ciklit). Në qoftë se kushti i ciklit plotësohet atëherë ekzekutohet *komanda* dhe i tërë procesi përsëritet. Përndryshe (kur kushti të mos përsëritet), procesi do të ndërpritet.

Për shembull, supozojmë se dëshirojmë të mbledhim të gjitha numrat nga numri 1 deri tek një numër i plotë i fundmë *n*, atëherë kjo mund të shprehet kështu:

```
int i = 1;
sum = 0;
while ( i <= n )
        sum += i++;
```

këtu vërejmë se **while** është një kusht ku brenda kllapave kemi shprehjen i <=n, pra kjo shprehje na tregon se komanda **sum +=i++;** do të ekzekutohet vetëm kur vlera e i të jetë më e vogël se vlera e n. Më poshtë kemi marrë vlerën e i = 5, dhe në tabelë kemi shpjeguar për të gjithë ciklet veç e veç:

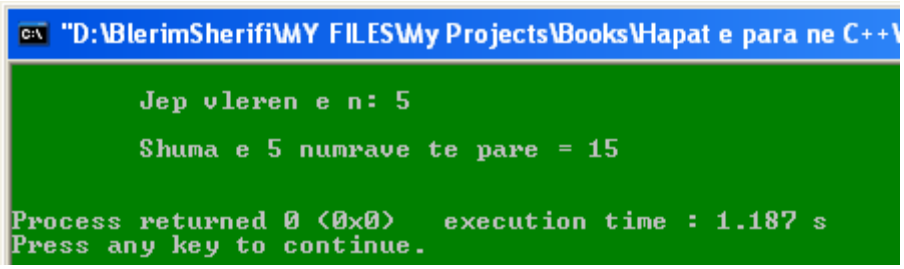
Cikli	i	n	I<=n	Sum + = i++
I parë	1	5	true	1
I dytë	2	5	true	3
I tretë	3	5	true	6
I katërt	4	5	true	10
I pestë	5	5	true	15
I gjashtë	6	5	false	

Kështu këtë shembull do ta kryejmë edhe përmes një programi në vazhdim.

Shembull: Të bëhet program që do të llogaris shumën e n numrave të parë natyrorë përmes deklaratës **while**. Vlera e n të jepet nga tastiera kurse rezultati në fund të nxirret në ekran.

```
# include <iostream>
using namespace std;
int main ()
{
   int n, i, shuma;
   cout << "\n\tJep vleren e n: ";
   cin >> n;
   i = 1;
   shuma = 0;
   while ( i <= n)            // deklarata while (kushti i ciklit while)
      shuma += i++;           // komanda që llogaritet kur të jetë plotësuar kushti

   cout << "\n\tShuma e " << n << " numrave te pare = " << shuma;
   cout << "\n\n";
   return 0;
}
```

Pas ekzekutimit të programit në qoftë se nga tastiera fusim numrin 5 atëherë rezultati do të duket njëjtë si në figurën e më poshtme:



Deklarata *do*

Kjo deklaratë (e thirrur edhe si cikli **do while**) sikurse edhe deklarata **while** përdoret për të krijuar cikël por dallon nga deklarata **while** se këtu në fillim ekzekutohet komanda, pastaj vlerësohet kushti. Forma e përgjithshme e kësaj komande duket kështu:

```
do
komanda;
while (shprehja);
```

Deklarata **do** është më pak e dobishme për të krijuar cikël. Kjo është më shumë e dobishme atëherë kur kemi vetëm një cikël për të kryer, pavarësisht nga kushti i ciklit. Për shembull ne mendojmë që në mënyrë të përsëritur të lexojmë një vlerë dhe ta shtypim atë, dhe ta ndalojmë kur vlera të jetë zero. Kjo mund të bëhet me anë të ciklit në vazhdim:

```
do {
cin >> n;
cout << n * n << '\n';
} while (n != 0);
```

Edhe këtu do të krijojmë këtë shembull në një program në *Code::Blocks* dhe do të shohim më mirë funksionin e deklaratës **do**.

Shembull: Të bëhet program ku do të fusim nga tastiera një numër dhe pasi të fusim numrin në ekran do të na nxirret ai numër i shumëzuar me vetveten. Kështu programi le të përsëritet deri sa vlera që ne do të fusim të jetë 0, dhe në këtë rast të ndërpritet cikli. Cikli të krijohet me anë të deklaratës **do**.

```
# include <iostream>
using namespace std;
int main()
{
   int n;
   cout << "\n\tCikli do filloi:\n\t";

   do {              // brenda katërkëndëshit me ngjyrë të kuqe kemi të hijezuar
      cin >> n;      // ciklin do i cili përsëritet deri sa vlera hyrëse e n të jetë zero.
      cout << "\t" << n * n << "\n\t";
     }while (n != 0);

   return 0;
}
```

Deklarata *for*

Komanda **for** (gjithashtu e thirrur edhe si cikli **for**) është e ngjashme me komandën **while**, por ka dy komponentë shtesë:

- Një shprehje e cila vlerësohet vetëm njëherë para çdo gjë tjetër, dhe
- Një shprehje që është vlerësuar në fund të çdo përsëritje (iteracioni).

Forma e përgjithshme e komandës **for** është kështu:

for (*shprehja1*; *shprehja2*; *shprehja3*)
***komanda*;**

Së pari vlerësohet *shprehja1*. Çdo herë rreth ciklit, *shprehja2* është ekzekutuar. Dhe nëse rezultati është jo zero atëherë *komanda* është ekzekutuar, dhe *shprehja3* është vlerësuar. Përndryshe cikli është ndërprerë.

Forma e përgjithshme e komandës **for** është ekuivalente me formën e përgjithshme të komandës **while**:

```
shprehja1;
while (shprehja2)
{
        komanda;
        shprehja3;
}
```

Përdorimi më i zakonshëm i ciklit **for** është për situata ku një ndryshore (variabël) ritet (*increment*) ose zvogëlohet (*decrement*) në çdo përsëritje të ciklit. Për shembull, cikli **for** në vazhdim kalkulon shumën e numrave natyror nga numri **1** deri tek numri **n**.

```
sum = 0;
for (i = 1; i <= n; ++i)
        sum += i;
```

Cikli funksionon në këtë mënyrë nëse vlerën e **n** e marrim **5**:

Cikli1: vlera fillestare e ndryshores **sum** është **0**, pastaj fillon cikli për **i = 1** (komponenti i parë i komandës for ekzekutohet njëherë para çdo gjë tjetër), nëse **i <=n** ekzekutohet *komanda* **sum += i;** dhe **sum** do të jetë **1** (**sum=sum+1**, ku vlera fillestare e **sum** është **0**). Pastaj kemi

komponentin e dytë ku një shprehje vlerësohet njëherë në fund të çdo përsëritje, dhe në këtë rast kemi shprehjen ++i, pra i ritet për 1 dhe bëhet 2. Tani përfunduam me *ciklin1*.

Cikli2: vlera e **i**-së tani është **2** dhe tani shprehja **i = 1;** nuk vlerësohet më sepse është vlerësuar njëherë në fillim të ciklit vetëm një herë (sipas definicionit të lartpërmendur). Vlerësohet *shprehja2* (nga forma e përgjithshme), në këtë rast kemi **i<=n;** pra nëse **i**-ja është më e vogël ose e barabartë me **n**, atëherë ekzekutohet *komanda* **sum +=i;** gjë që është e njëjtë me **sum=sum+i;** ku vlera e ndryshores **sum** do të bëhet **3** (**sum = 2 + 1**). Pastaj edhe në këtë cikël ekzekutohet *shprehja3* (nga forma e përgjithshme) në këtë rast **i++**. Kështu *cikli2* përfundon.

Cikli3: vlera e **i**-së është **3** (pra është rritur në fund të ciklit2 nga shprehja **i++**). Vlerësohet *shprehja2*, ku **i<=n** përkatësisht **3<=5** është **true** (plotësohet kushti), atëherë ekzekutohet *komanda* **sum+=i;** dhe vlera e **sum** bëhet **6**. Pastaj vlerësohet *shprehja3*. Dhe përfundon *cikli3*.

Cikli4: tani vlera e **i**-së është **4**. Vlerësohet *shprehja2*, ku krahasohet nëse **i<=n**, pra **4<=5** kjo jep **true** (e saktë) dhe ekzekutohet **komanda sum+=i;** ku **sum** bëhet **10** (**sum = 6 + 4**). Pastaj në fund të ciklit vlerësohet *shprehja3* përkatësisht **i++** (ritet **i**-ja për **1**). *Cikli4* përfundon.

Cikli5: tani vlera e **i**-së është **5**. Vlerësohet *shprehja2*, ku krahasohet nëse **i<=n**, pra **5<=5** kjo jep **true** pasi **5 = 5** dhe ekzekutohet *komanda* **sum+=i;** ku **sum** bëhet **15** (**sum=10+5**). *Cikli5* këtu përfundon.

Cikli6: tani vlera e **i**-së është **6**. Vlerësohet *shprehja2*, ku krahasohet nëse **i<=n**, pra **6<=5** e cila jep **false** (nuk plotësohet pasi **6** është më e madhe se **5**) dhe cikli **for** këtu mbaron, pra kompajleri del nga cikli. Në *ciklin6* nuk ekzekutohet *komanda* as *shprehja3* por thjesht komplet cikli **for** mbaron.

C++ gjithashtu lejon që **shprehja1** të jetë edhe variabël e definuar për shembull:

```
for (int i = 1; i<=n; i++)
        sum+=i;
```

kjo mund të duket edhe kështu:

```
int i;
for (i = 1; i<=n; i++)
```

```
sum+=i;
```

Po ashtu ndonjëri nga tre shprehjet në ciklin **for** mund të jenë të zbrazët. Ja një shembull ku do të fshijmë shprehjen1 dhe shprehjen3 dhe do të fitojmë një cikël **for** ekuivalent me një cikël **while**:

```
for (; i!=0 ;)
        komanda;        // është ekuivalente me
while (i!=0)
        komanda;
```

Në qoftë se i fshijmë të tre shprehjet atëherë komanda **for** do të na jep neve një cikël të pafundmë. Ja një shembull:

```
for (; ; )                  // cikël i pafundmë
        komanda;
```

Gjithashtu me komandën **for** ne kemi mundësi të krijojmë cikle me shumë ndryshore të ciklit, gjë që nuk është e pazakontë. Në këtë rast, ndryshoret i ndajmë me presje:

```
for (i = 0, j = 0; i + j < n; ++i, ++j)
        komanda;
```

Pasi që for është komandë atëherë mund që të përdorim brenda cikleve tjera, pra do të kemi komandë brenda komande. Për shembull:

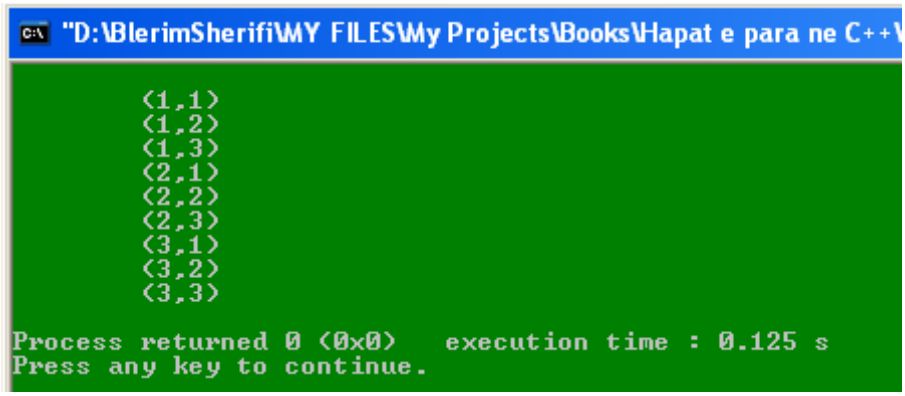
```
for (int i = 1; i <= 3; ++i)
        for (int j = 1; j <= 3; ++j)
        cout << '(' << i << ',' << j << ")\n";
```

ja më poshtë do të krijojmë një program ku do të kemi këtë shembull:

Shembull: Të krijohet program ku do të kemi dy cikle **for** njëri brenda tjetrit. Te cikli i parë *shprehja1* të jetë: **int i = 1;** *shprehja2*: **i<=3;** dhe *shprehja3*: **++i**. Kurse cikli i dytë të jetë brenda ciklit të parë dhe shprehja1 të jetë: **int j = 1;** *shprehja2*: **i<=3;** dhe *shprehja3*: **++j**. Në fund rezultati të nxirret në ekran kështu: **(i,j)**.

```
# include <iostream>
using namespace std;
int main ()
{
   cout << "\n\t";
   for (int i = 1; i<=3; i++)
      for (int j = 1; j<=3; j++)
         cout << '(' << i << ',' << j << ")\n\t";

   return 0;
}
```

Pas ekzekutimit të programit rezultati në ekran do të duket kështu:



Deklarata *range-based for*

Kjo deklaratë ekzekuton deklaratën e përsëritur dhe sekuenciale për çdo element në shprehje. Gjithashtu përdoret për ndërtimin e cikleve që patjetër duhet ekzekutuar nëpërmjet një "kufiri", e cila është definuar si çdo gjë që mund të përsërisim përmes. Përshembull, *std::vector*, apo ndonjë tjetër sekuencë STL kufiri i të cilit është përcaktuar nga *begin()* dhe *end()*. Emri që është deklaruar në pjesën *for-range-declaration* është lokale për deklaratën **for** dhe nuk mund të ri-deklarohet në *shprehje* apo *deklaratë*.

Shembulli:

Deklarata *continue*

Komanda **continue** shërben për të përfunduar ciklin që është duke u ekzekutuar dhe në vend kalon në ciklin e ardhshëm.

Tek cikli **while** dhe **do**, përsëritja e ardhshme fillon nga kushti i ciklit. Në ciklin **for** përsëritja e ardhshme fillon nga *shprehja3* e ciklit. Për shembull, cikli i cili në mënyrë të përsëritur lexon një numër nga tastiera, kalkulon atë por i anashkalon numrat negativ, dhe përfundon kur numri i futur të jetë zero, mund të shprehet kështu:

```
do {
cin >> num;
if (num < 0) continue;
        num+=1;
} while (num != 0);
```

Kjo është ekuivalente me:

```
do {
cin >> num;
if (num >= 0) {
        num +=1;
}
        } while (num != 0);
```

Një variant i ciklit që lexon një numër deri në **n** herë, të shprehur me komandën **for**:

```
for (i = 0; i < n; ++i) {
cin >> num;
if (num < 0) continue;          // kërcen te :    ++i
        num+=1;
}
```

Kur kemi cikël brenda ciklit, atëherë komanda **continue** merret vetëm për ciklin që ndodhet brenda. Për shembull, në kodin në vazhdim komanda **continue** vlen vetëm për ciklin **for** e jo edhe për ciklin **while**:

```
while (kushti)
{
        for (i = 0; i < n; ++i)
        {
                cin >> num;
        if (num < 0) continue; // dmth kërcen tek: ++i
```

```
                num +=1;
            }
            komanda;
        }
```

Komanda break

Komanda **break** mund të shfaqet brenda ciklit (*while, do, ose for*) ose brenda komandës ***switch***. Kjo shkakton kërcim jashtë ciklit të këtyre komandave, dhe shkakton përfundimin e tyre. Pra kjo komandë dallon nga komanda *continue* sepse nuk kalon në ciklin e ardhshëm por ndalon tërë procesin e përsëritjes (ciklit) menjëherë. Ashtu si edhe komanda *continue* edhe kjo komandë përdoret brenda cikleve dhe komandës *switch*, gjë që do të ishte një gabim po që se përdoret diku jashtë këtyre komandave.

Për shembull marrim një log in ku përdoruesi fut username dhe paswordin.

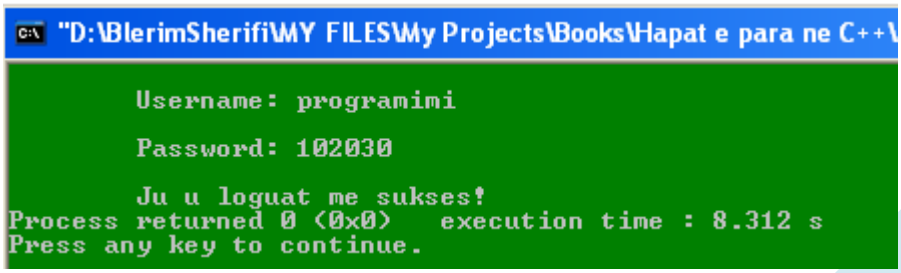
```
for (int i = 1; i<=3; i ++)
{
        cout << "Username: ";
        cin >> username;
        if (username == "programimi")
        {
                cout <<"Password:" ";
                cin >> password;
        }
        if (password != "102030")   //nëse passwordi është gabim
                break;          //komanda break e ndërprenë ciklin
                cout << "Gabim provo perseri.";
}
```

Për ta kuptuar më mirë kodin e lartshënuar, në vazhdim kemi shembullin e radhës ku kemi krijuar një program për log in.

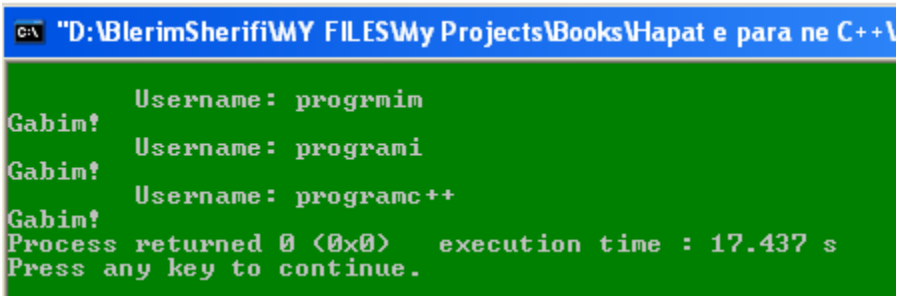
Shembull: Të bëhet program për një log in. Të krijohen dy stringje username dhe password, ku përdoruesi do të fus usernamin dhe passwordin. Në qoftë se përdoruesi jep 3 herë të dhënat gabim, qasja e tij të ndërpritet dhe të nxirret në ekran mesazhi "Gabim!".

```
# include <iostream>
using namespace std;
int main ()
{
   string username;
   string password;
   int i;
   for (i=1; i<=3; i++)
   {
      cout << "\n\tUsername: ";
      cin >> username;
      if (username=="programimi")
      {
         cout << "\n\tPassword: ";
         cin >> password;
      }
      if (password=="102030")
      {
         cout << "\n\tJu u loguat me sukses!";
         break;
      }
      cout << "Gabim!";
   }
   return 0;
}
```

Pas ekzekutimit te programit rezultati ne ekran do të duket kështu nëse të dhënat i japim të sakta:



por nëse të dhënat i japim gabim atëherë rezultati do të duket kështu:



Komanda goto

Komanda goto siguron nivelin më të ulët të kërcimit (ang. jumping). Kjo komandë na mundëson që të kërcejmë nëpër program nga një pjesë e kodit në një tjetër. Forma e përgjithshme e kësaj komande duket kështu:

goto *label*;	*// goto është tipi, kurse label shkruhet një label ku do të //kërcen programi*
***label*:**	*// ndërsa kjo shprehje vendoset në vendin ku duam të //kërcen komanda **goto**.*

Shembull marrim një program ku në të do të kemi tre reshta për ekzekutim pra do të nxjerrim në dalje tre fjali. Pas fjalisë së dytë do të vendosim komandën **goto** e cila do ta kalon këtë resht dhe do të fillon te reshti i tretë.

```
cout << "Ky eshte reshti i pare.";
goto treta;
cout << "Ky eshte reshti i dyte.";
treta:
cout << "Ky eshte reshti i trete.";
```

pra siç shihet edhe tek kodi, programi do të ekzekuton reshtin e parë pastaj komanda **goto** kërcen tek labeli *"treta:"* ku përsëri fillon ekzekutimi dhe do të ekzekutohet reshti i tret. Reshti i dytë nuk

ekzekutohet pasi atë e kalon komanda **goto**. Në fund nëse e ekzekutojmë këtë kod në ekran do të na paraqitet vetëm reshti i parë dhe i tretë.

Po ashtu vlen të theksohet se kjo me anë të *goto* ne mund të kalojmë edhe në fillim të kodit, pra me anë të kësaj komande mund të krijojmë edhe cikël, por kjo gjë nuk është e përshtatshme pasi për cikël kemi komandat tjera më të përshtatshme. Gjithashtu kjo komandë shpeshherë përdoret edhe për dalje nga cikli. Në programin e mësipërm (log in) do të përdorim komandën *goto*, pra do të del nga cikli.

Shembull: Të bëhet program për një log in. Të krijohen dy stringle username dhe password, ku përdoruesi do të fus usernamin dhe passwordin. Në qoftë se përdoruesi jep 3 herë jep të dhënat gabim, qasja e tij të ndërpritet dhe të nxirret në ekran mesazhi "Gabim!". **Të përdoret komanda goto.**

```
# include <iostream>
using namespace std;
int main ()
{
   string username;
   string password;
   int i;
   for (i=1; i<=3; i++)
   {
      cout << "\n\tUsername: ";
      cin >> username;
      if (username=="programimi")
      {
         cout << "\n\tPassword: ";
         cin >> password;
      }
      if (password=="102030")
      {
         cout << "\n\tJu u loguat me sukses!";
                  goto dil;          // kur passwordi të jetë saktë goto shkon tek dil:
                                        pra del jashtë ciklit
      }
      cout << "Gabim!";
   } dil:
   return 0;
}
```

Pasi komanda goto na mundëson kërcim të lirë nëpër program (kërcim të pa strukturuar), ajo gjithashtu lehtë mund edhe të keqpërdoret. Shumica e programorëve përdorin këtë komandë në favor të programimit të qartë.

Komanda return

Komanda **return** kryesisht përdoret tek funksionet ku ju mundëson atyre të kthejnë vlerë tek thirrësi i tyre. Forma e përgjithshme e kësaj komande është:

return *shprehja*;

ku *shprehja* tregon vlerën e kthyer nga funksioni. Është e patjetërsueshme që lloji i kësaj shprehje të jetë në përputhshmëri me tipin e funksioni, për shembull nëse funksioni është deklaruar i tipit **int** edhe shprehja duhet të jetë e tipit **int**. Gjithashtu tek funksionet që janë të deklaruar të tipit **void** komanda **return** duhet të jetë e zbrazët (**empty**), kështu:

return;

Funksioni i vetëm për të cilin kemi diskutuar deri më tani është funksioni **main**. Lloji i kthimit i këtij funksioni është gjithmonë **int**. Vlera kthyese e funksionit **main** është se ç'far programi kthen tek sistemi operativ kur përfundon ekzekutimin e tij. Sipas UNIX, programi kthen 0 kur ka përfunduar me sukses. Ndërsa kthen kodin e gabimit (**erorr code**) kur ka gabime në program. Për shembull:

```
int main ()          // funksioni main, tipi int pasi kthen numrin 0
{                    // fillimi i  trupit main
        cout << "Hello World\n";
        return 0;        // kthimi i numrit 0, pra zbrazet memoria
}                        // mbarimi i trupit main
```

Shembuj

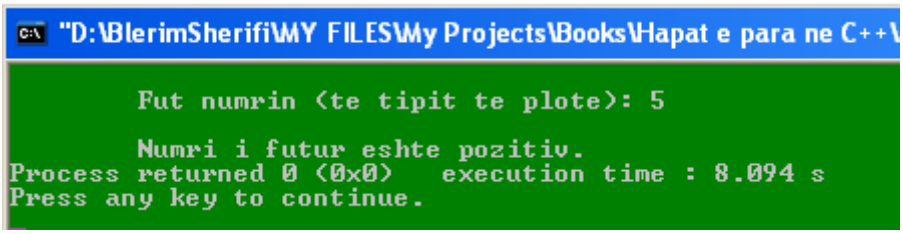
Në këtë kapituj keni të ilustruar disa shembuj të zgjidhur dhe disa të pa zgjidhur.

Shembull1: Të bëhet program ku si hyrje do të futet një numër i plotë dhe më pas të krahasohet nëse është pozitiv apo negativ. Të nxirret mesazh njoftimi në ekran.

Zgjidhje:

```
#include <iostream>
using namespace std;
int main ()
{
   int numri;
   cout << "\n\tFut numrin (te tipit te plote): “;
   cin >> numri;
   if (numri > 0)
         cout << “\n\tNumri i futur eshte pozitiv.”;
   else cout << “\n\tNumri i futur eshte negativ.”;
   return 0;
}
```

Pas ekzekutimit dalja në ekran do të jetë:

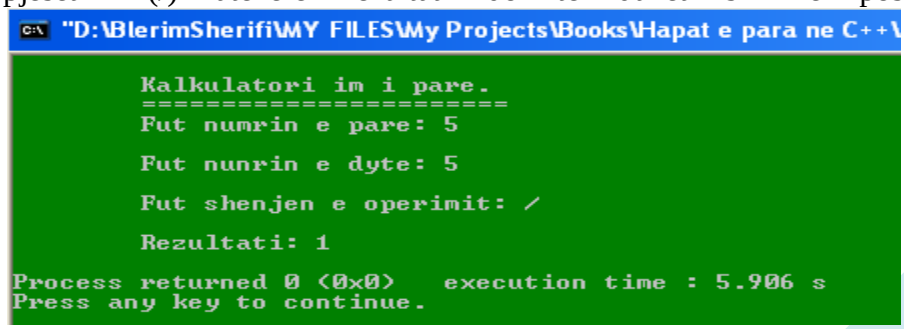


Shembull2: Të bëhet program, një kalkulator i thjeshtë i cili do të kryejë 4 operacionet themelore matematikore (+, -, *, /). Në fillim të jepen numrat me të cilët do të operohet dhe më pas të futet shenja e operimit. Varësisht nga shenja e futur të kryhet operacioni mbi numrat. Dalja të shfaqet në ekran. Të bëhet përmes komandës **switch**.

Zgjidhje:

```
# include <iostream>
using namespace std;
int main ()
{
   int a,b,L;
   char s;
   cout <<"\n\tKalkulatori im i pare.";
   cout <<"\n\t======================";
   cout << "\n\tFut numrin e pare: ";
   cin >> a;
   cout << "\n\tFut nunrin e dyte: ";
   cin >> b;
   cout << "\n\tFut shenjen e operimit: ";
   cin >> s;

   switch (s)
   {
      case '+': L=a+b;
      break;
      case '-': L=a-b;
      break;
      case '*': L=a*b;
      break;
      case '/': L=a/b;
      break;
   }
   cout << "\n\tRezultati: " << L << endl;
   return 0;
}
```

Pas ekzekutimit të programit, nëse numrat e futur i japim 5 dhe operatorin pjesëtim (/) atëherë rezultati do të duket si më poshtë:

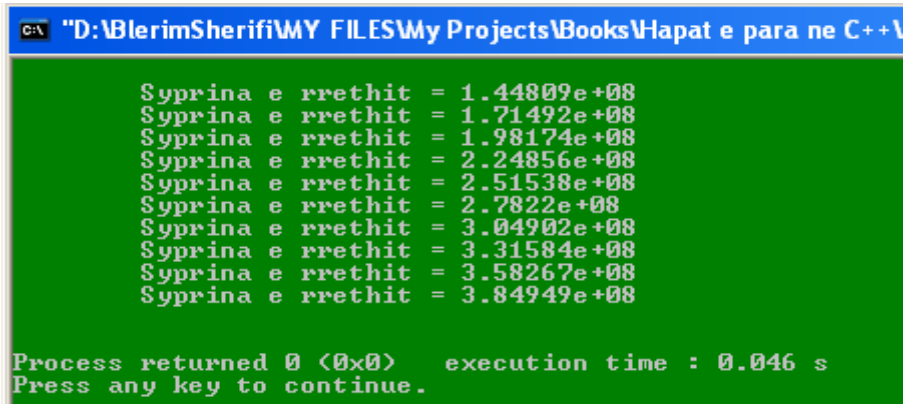


Shembull3: Të bëhet program ku do të llogaritet syprina e rrethit 10 herë. Për çdo herë të nxirret në ekran rezultati "*Syprina e rrethit = 12.11*". Vlera fillestare e r të jetë 1 pastaj për çdo përsëritje të ritet për 1. Përsëritja të realizohet me anë të komandës While.

Zgjidhje:

```
# include <iostream>
using namespace std;
int main ()
{
   double const pi = 3.1415;
   int r,i;
   double S;
   i = 1;
   while (i<=10)
   {
      S = r * r * pi;
      cout <<"\n\tSyprina e rrethit = "
          << S;
      r +=1;
      i++;
   }
   cout << "\n\n";
   return 0;
}
```

Pas ekzekutimit të programit rezultati në ekran do të duket kështu:

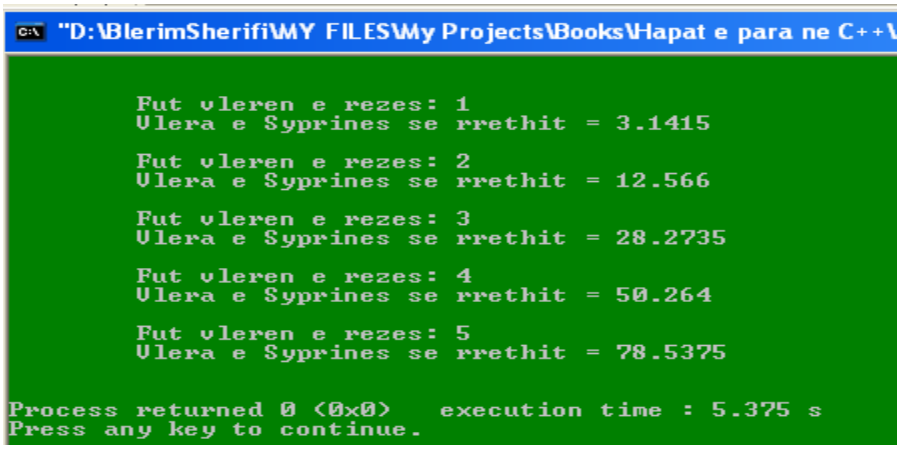


Shembull4: Të bëhet program ku do të llogaritet syprina e rrethit 5 herë. Për çdo herë të nxirret në ekran rezultati "*Syprina e rrethit = 12.11*". Vlera e r të futet nga tastiera. Përsëritja të bëhet me anë të komandës **do while**.

Zgjidhje:

```
# include <iostream>
using namespace std;
int main ()
{
   double const pi = 3.1415;
   int r,i=1;
   double S;
   do
   {
      cout << "\n\n\tFut vleren e rezes: ";
      cin >> r;
      S = r * r * pi;
      cout << "\tVlera e Syprines se rrethit = "
           << S;
      ++i;
   }
   while (i<=5);
   cout << "\n\n";
   return 0;
}
```

Pas ekzekutimit të programit nëse vlerat e rezes i fusim nga numri 1 deri 5, atëherë rezultati në ekran do të shfaqet si në figurën e më poshtme.



Shembull5: Të bëhet program ku do të nxirret në ekran 10 herë stringu "Hap pas hapi C + +!", ku para këtij stringu të ketë numra, pra nga 1 deri në 10.

Shembull:

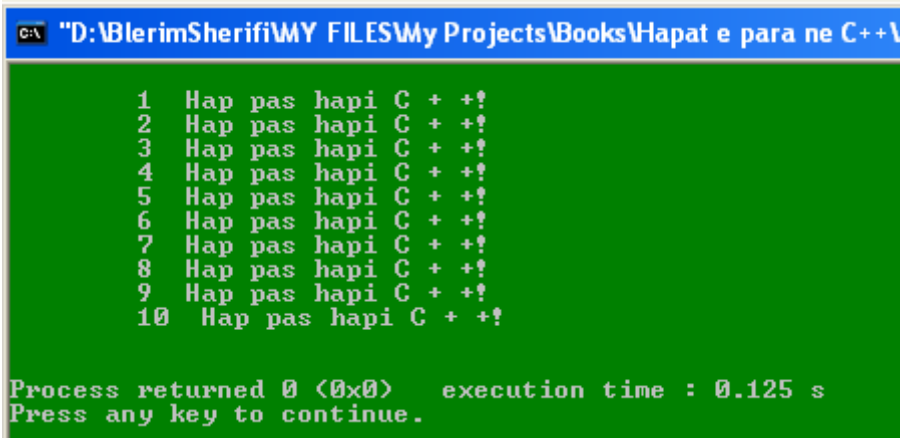
1. Hap pas hapi C + +!

. . . .

Zgjidhje:

```
# include <iostream>
using namespace std;
int main ()
{
   int i;
   string a;
   a = "Hap pas hapi C + +!";

   for (i=0; i<10; i++)
   {
      cout << "\n\t" << i + 1
           << "  "
           << a;
   }
   cout << "\n\n";
   return 0;
}
```

Pas ekzekutimit të programit në ekran do të na shfaqet ky rezultat:



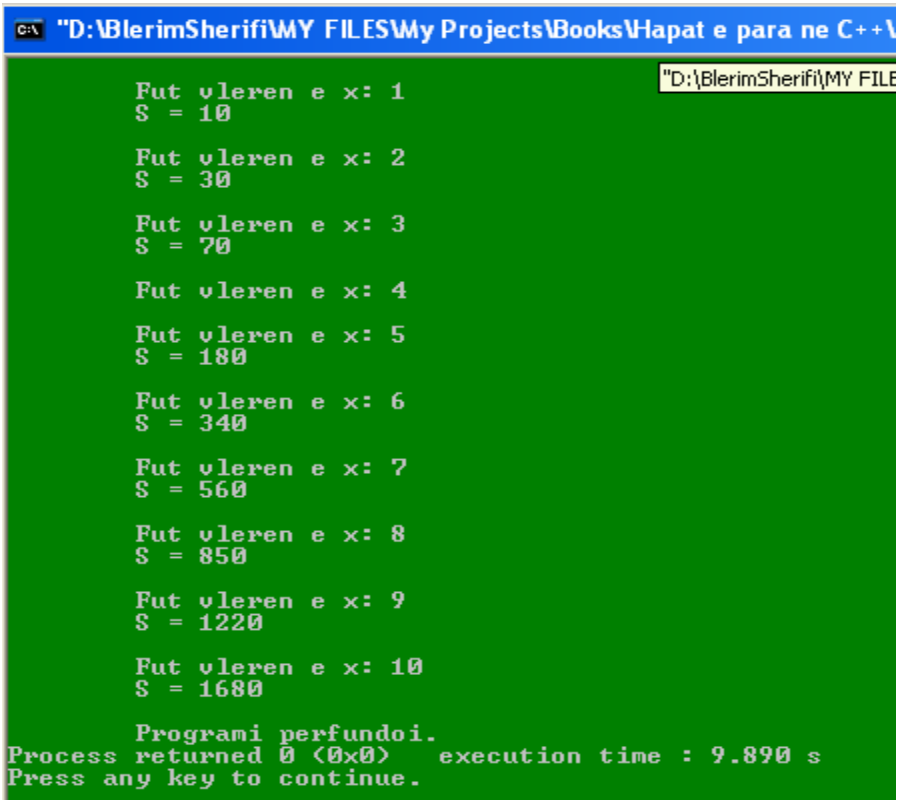
Shembull6: Të bëhet program ku do të llogaritjet kjo formulë matematikore:

Vlera e variabëlit x të futet nga tastiera, dhe në qoftë se është më e vogël se 0 të ndërpritet cikli. Në program të përdoren komandat: **for, continue, break**.

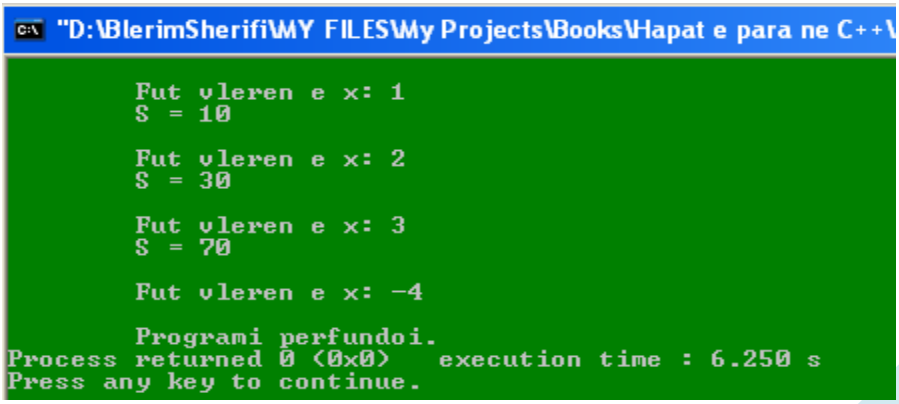
Zgjidhje:

```
# include <iostream>
using namespace std;

int main ()
{
   int i,x,S,n;
   n = 10;
   S = 0;

   for (i=0;i<n;i++)                          // cikli for
   {
      cout << "\n\tFut vleren e x: ";
      cin >> x;
      if (x<0) break;                  //komanda break nëse x është negativ
      else if (x==4) continue;         //komanda continue nëse x është 4 kërcen ciklin
      else                        // nëse x është i lejuar ekzekutohen komandat në vazhdim
      {
         S += 5 * x * i + 10;
         cout <<"\tS = "
             << S << endl;
      }
   }
   cout << "\n\tPrograma perfundoi.";
   return 0;
}
```

Pas ekzekutimit të programit në ekran dalja do të jetë:



- siç shihet në këtë figurë ku vlera e futur është 4 cikli ekzistues kalon në ciklin e ardhshëm përmes komandës *continue*. Kurse në rastet tjera llogaritet vlera e **S** dhe nxirret në ekran njëkohësisht.
- Por nëse ndonjëra vlerë e futur x është negative atëherë cikli ndërpritet dhe përfundon programi. Këtë gjë e vërejmë në figurën e mëposhtme:



Vërejtje! – *Ndonëse këto shembuj nuk janë të vlefshëm apo që mund të ndihmojnë apo të përdoren plotësisht për zgjedhje të ndonjë problemi të caktuar, megjithëse zgjedhin ndonjë pjesë të tillë, por qëllimi kryesorë është që të shohim përdorimin e komandave continue dhe breake, praktikisht në program. Kjo vlen edhe për shembujt paraprak. Ndërsa për zgjidhje të plotë të ndonjë problemi të caktuar do të shohim në fund të librit në kapitull të veçantë për zgjedhje të problemeve të ndryshme.*

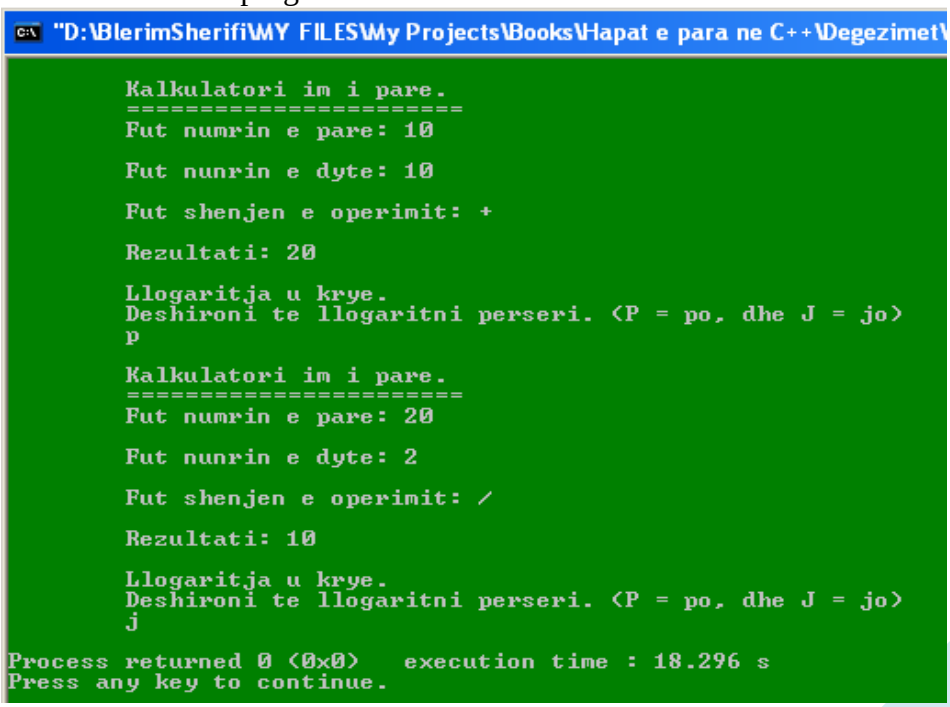
Shembull7: Të bëhet program një kalkulator i thjeshtë (si te shembulli 2), por me ndryshimin e vetëm ku në fund të programit të shfaqet ky mesazh *"Llogaritja u krye. Dëshironi të llogaritni përsëri. (P – po, dhe J – jo)"*. Dhe nëse përdoruesi shtyp nga tastiera tastin P të përsëritet programi, kurse nëse shtyp J të ndërpritet. Të përdoret komanda **goto**.

Zgjidhje:

```
# include <iostream>
using namespace std;
int main ()
{
   int a,b,L;
   char s,f;

   fillimi:                                   //shtegu ku kërcen goto me label fillimi
   cout <<"\n\tKalkulatori im i pare.";
   cout <<"\n\t=======================";
   cout << "\n\tFut numrin e pare: ";
   cin >> a;
   cout << "\n\tFut nunrin e dyte: ";
   cin >> b;
   cout << "\n\tFut shenjen e operimit: ";
   cin >> s;
   switch (s)
   {
      case '+': L=a+b;
      break;
case '-': L=a-b;
      break;
```

```
        case '*': L=a*b;
        break;
        case '/': L=a/b;
        break;
    }
    cout << "\n\tRezultati: " << L << endl;
    cout << "\n\tLlogaritja u krye."
         << "\nDeshironi te llogaritni perseri. (P = po, dhe J = jo)\n\t";
    perseri:                          //shtegu ku kërcen goto me label perseri
    cin >> f;
    if (f == 'p' || f == 'P') goto fillimi;
    else if (f == 'j' || f == 'J') goto fundi;
    else
    {
        cout << "\n\tKarakter i gabuar!";
        goto perseri;
    }
    fundi:                            //shtegu ku kërcen goto me label fundi
    return 0;
}
```

Pas ekzekutimit të programit do të shohim këtë rezultat:



- Pra siç shihet edhe te figura e sipërme në fillim mbledhim dy numra pastaj pasi na shfaqet rezultati i atyre numrave pas mesazhit i cili na tregon e *"Llogaritja u krye."*, na shfaqet edhe një mesazh tjetër ku thotë *"Deshironi te llogaritni perseri. (P = po, dhe J = jo)"*, ku në fakt na ofron një mundësi zgjedhje në këtë programin tonë ku nëse ne e zgjedhim alternativën e parë do të përsëritet programi nga fillimi dhe nëse e zgjedhim alternativën e dytë programi do të mbyllet. Kështu në figurë pas një llogaritje kemi zgjedhur që të bëjmë edhe një llogaritje tjetër kështu kem i dhënë alternativën **p** e cila do të thotë Po, dhe pas llogaritjes së dytë kemi zgjedhur alternativën J e cila do të thotë Jo dhe programi të mbyllet. Tek alternativa vërejmë se shkronjat janë të mëdha por ne kemi futur shkronjat e vogla, kjo është bërë e mundur falë këtij kushti:

```
if (f == 'p' || f == 'P') goto fillimi;
else if (f == 'j' || f == 'J') goto fundi;
```

pra tek këto dy reshta ne e bëjmë të mundur për shfrytëzuesin që pa varësisht se a fut shkronjë të madhe apo të vogël të vlejë programi. Për këtë na kanë ndihmuar operatorët e relacionit ku me operatorin || nënkuptojmë ose, pra, nëse **f** është e barabartë me **p** ose me **P** të vazhdojë programi.

- Gjithashtu lind pyetja se nëse në vend të shkronjës **p** dhe **j** fusim ndonjë shkronjë tjetër atëherë ç'far do të ndodh me programin tonë? Atëherë, për këtë pyetje kujdeset ky kod në vazhdim:

```
else
{
        cout << "\n\tKarakter i gabuar!";
        goto perseri;
}
```

pra me anë të këtij kodi ne kujdesemi që mos të futet karaktere të padëshiruara. Ky kod funksionon në këtë mënyrë, pra është vazhdim i kushteve paraprake të alternativës 1 dhe 2 dhe në këtë rast na paraqitet si alternativë e tretë por e gabuar ku nëse ne fusim ndonjë karakter të ndryshëm nga p dhe j atëherë kjo pjesë e programit do të nxjerr një mesazh njoftimi për gabimin e bërë dhe do të na kthen prapa tek zgjedhja e alternativës. Pra këtë na mundëson komanda (**goto perseri;)**.

Kontrollim Njohurish

Në këtë kapituj do të keni disa detyra të pa zgjidhura. Këto detyra janë kryesisht nga ky kapitull, ku vlerësohen si përsëritje apo kontroll e njohurive të marra nga studentit nga ky kapitull.

1. Çka janë degëzimet dhe për ç'far shërbejnë ato?

2. Sa lloje të degëzimeve kemi?

3. Cila është dalja e kodit në vazhdim po që se:

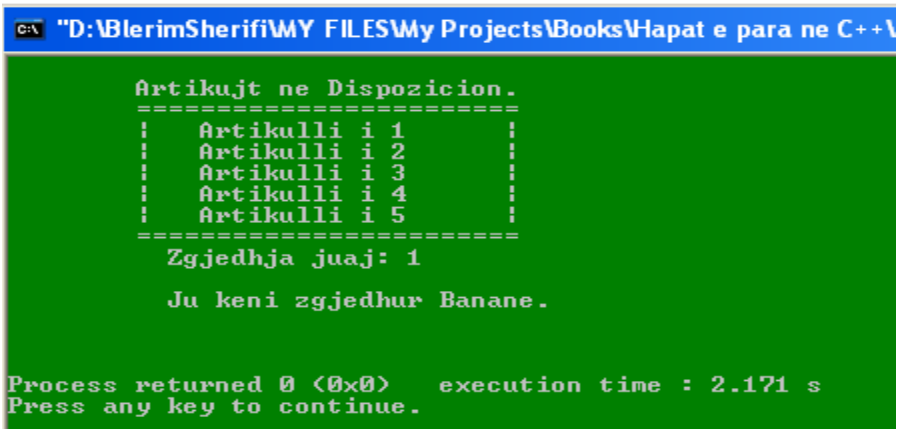
```
int a;
if (a > 0)
        cout << "Numër pozitiv.";
else
        cout << "Numër negativ.";
```

a) a = 5 b) a = -2 c) a = 0

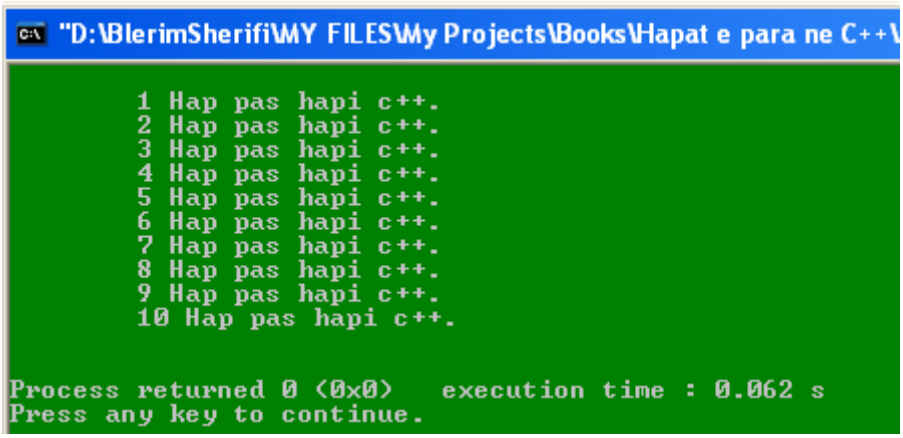
4. Të bëhet program ku do të nxirren në ekran 5 artikuj ushqimor, dhe varësisht se cilin numër e zgjedh të shfaqet në ekran artikulli me numër përkatës. Të përdoret komanda **switch** për zgjedhjen e alternativave.

Artikujt	Emrat
Artikulli 1	Banane
Artikulli 2	Molle
Artikulli 3	Dardhe
Artikulli 4	Kivi
Artikulli 5	Pjeshkë

Për shembull nëse nga tastiera fusim numrin 1 rezultati do të duket kështu:



5. Të bëhet program i cili do të nxjerr në ekran 10 herë tekstin “Hap pas hapi c++”. Programi të bëhet me ciklin while.
   Programi të duket kështu:



6. Cila është dalja e kodit në vazhdim:

```
do
{
        A +=B;
        B = i;
        A = B + 1;
        cout << "\n\tA = " << A;
        cout << "\n\tB = " << B;
        ++i;
} while (i<5);
```

a)

```
A = 4
B = 1
A = 5
B = 2
A = 6
B = 3
A = 7
B = 4
```

b)

```
A = 3
B = 1
A = 4
B = 2
A = 5
B = 3
A = 6
B = 4
```

c)

```
A = 2
B = 1
A = 3
B = 2
A = 4
B = 3
A = 5
B = 4
```

7. Ku është dallimi në mes komandës **for** dhe **while**?

8. Të llogaritet seria gjeometrike e dhënë: — përmes ciklit **for**.

9. Cila është dalja e kodit në vazhdim:

```
fillimi:
cout << "Mire se vini ne C++." << endl;
goto fillimi;
```

10. Të ndërtohet program i cili do të llogarit këtë vëllimi i cilindrit: , dhe rezultati të shfaqet në ekran, ku pas rezultatit ti jepet mundësia shfrytëzuesit se a dëshiron të përsëris programin edhe njëherë nëse por të futet karakteri P dhe nëse jo të futet karakteri J.

Kapitulli VI

Funksionet

Nëntitujt:

- ✓ **Një funksion i thjeshtë**
- ✓ **Llojet e funksioneve**
- ✓ **Shtrirja globale dhe locale**
- ✓ **Operatori i fushës (shtrirjes)**
- ✓ **Ndryshoret automatike (auto)**
- ✓ **Ndryshoret regjistruese (register variables)**
- ✓ **Ndryshoret dhe funksionet statike**
- ✓ **Ndryshoret dhe funksionet eksterne**
- ✓ **Konstantet simbolike**
- ✓ **Numëruesit**
- ✓ **Runtime Stack**
- ✓ **Funksionet Inline**
- ✓ **Rekursivi**
- ✓ **Argumentet përmes komandave në reshta**
- ✓ **Shembuj**
- ✓ **Kontrollo Njohurit**

Funksionet

Ky kapitull përshkruan funksionet e definuar nga përdoruesi si blloqe kryesore për C++ programet. Deri më tani funksioni i vetëm që kemi përdorur është ai kryesor (main), pra sikurse edhe funksioni kryesor ashtu edhe funksionet që ne vet i krijojmë do të kenë pamjen e njëjtë.

Një funksion jep një mënyrë të përshtatshme për paketimin e mjeteve llogaritëse. Kështu që mund të përdoret sa herë që nevojitet. Një funksion i definuar përbëhet nga dy pjesë:

1. Interfejsi (Ndërfaqja) - Interfejsi i një funksioni (gjithashtu i thirrur edhe si prototipi i funksionit), saktëson se si ai mund të përdoret. Prototipi përbëhet nga tre entitete:
    - **Emri** i funksionit. Ky është thjeshtë identifikuesi unik.
    - **Parametrat** e funksionit (gjithashtu thirren signature simbole). Ku mund të deklarohen zero, një apo më shumë parametra që shërbejnë për kalimin e vlerave në dhe prej funksionit.
    - **Kthimi i llojit** të funksionit. Kjo specifikon llojin e kthimit të vlerës së funksionit. Funksionet që nuk kthejnë as një vlerë duhet të kenë tipin e kthimit **void**.
2. Trupi i funksionit përmban hapat e llogaritjes (komandat ang. statemants) që e përbëjnë funksionin.

Duke përdorur një funksion përfshihet edhe "thirrja" e tij. Thirrja e funksionit përbëhet nga emri i funksionit i ndjekur nga operatori thirrës i kllapave '()', ku brenda përmban zero ose më shumë argumente të ndara me presje. Numri i argumenteve duhet të përputhet me numrin e parametrave të funksionit. Po ashtu edhe tipi i argumentit në radhë duhet të jetë i njëjtë me tipin e parametrit të prototipit në radhë. Kur thirrja e funksionit ekzekutohet, argumentet vlerësohen në fillim dhe vlera e tyre caktohen për parametrat përkatës. Pastaj ekzekutohet trupi i funksionit. Në fund, funksioni kthen vlerën (nëse ka) tek thirrës.

Një thirrje e një funksioni i cili ka vlerë të kthimit "non-void" (jo-të pakthyeshëm) jep vlerë kthyese, thirrja është një shprehje dhe mund të përdoret në shprehje tjera. Në anën tjetër një thirrje e një funksioni vlera e kthimit e të cilit është void (pakthyeshëm) është vetëm një deklaratë (komanda).

Një funksion i thjeshtë

Në vazhdim do të paraqesim një funksion të thjeshtë i cili nuk ka vlerë kthyese pra funksion të tipit **void**. Funksioni bën shtypjen e tekstit *"Funksioni im i pare."* në ekran, ngjashëm sikurse edhe tek programi i parë në kapitullin 1.

Shembull1: Të bëhet program ku do të krijohet funksioni me emrin "funksioni" i tipit void, ku brenda tij të shtypet teksti "Funksioni im i pare.". Thirrja e funksionit të bëhet nga programi **main**.

```
# include <iostream>
using namespace std;

void funksioni()                    // deklarimi i funksionit
{                                   //fillimi i trupit të funksionit
   cout << "\n\tFunksioni im i pare." << endl;
}                                   //mbarimi i trupit të funksionit
int main()
{
      funksioni();                  //thirrja e funksionit brenda trupit main

   return 0;
}
```

Shënim:

Tek ky program shohim se funksionin e kemi deklaruar para funksionit **main** në këtë rast nuk e shënojmë prototipin e funksionit, por nëse ai shënohet në fund të funksionit **main** atëherë në fillim duhet patjetër të shënohet prototipi. Po ta shënonim funksionin tonë në shembullin e mësipërm në fund të funksionit kryesor (main) atëherë prototipi i tij do të ishte:

void funksioni();

Po ashtu nga Shembulli1 shohim se tek funksioni nuk kemi parametra, pra, brenda kllapave '**()**' nuk ka as një parametër të deklaruar, gjithashtu tipi i funksionit është **void** pra nuk ka vlerë kthyese.

Thirrja e funksionit bëhet brenda funksionit kryesorë (*main*) me anë të kësaj komande:

funksioni();

pra për thirrjen e funksioneve pa vlerë kthyese në fillim shënohet emri i funksionit më pas operatori **'()'** i kllapave.

Llojet e funksioneve

Kemi këto lloje të funksioneve:

- **Funksione pa vlerë kthyese – funksion ku nuk kthen vlerë:**
  - *Funksione pa vlerë kthyese pa parametra, dhe*
  - *Funksione pa vlerë kthyese me parametra.*
- **Funksione me vlerë kthyese – funksion ku kthen vlerë tek thirrësi i tij:**
  - *Funksione me vlerë kthyese pa parametra, dhe*
  - *Funksione me vlerë kthyese me parametra.*

Funksione pa vlerë kthyese

Funksione pa vlerë kthyese do të thotë se funksioni në fjalë nuk kthen vlerë tek thirrësi i tij. Funksionet e tilla quhen *funksione të tipit* ***void***. Këto funksione edhe pse nuk kanë vlerë kthyese nuk do të thotë se nuk mund të pranojnë vlera hyrëse (parametra). Kështu mund të themi se këto funksione ndahen në:

- *Funksione pa vlerë kthyese pa parametra, dhe*
- *Funksione pa vlerë kthyese me parametra.*

Funksione pa vlerë kthyese pa parametra

Këto funksione nuk përmbajnë parametra, pra kur thirren ato në programin kryesorë apo nga vendi ku thirren nuk marrin parametra.
Në shembullin në vazhdim do të kemi një funksion **void** pa parametra.

Shembull2: Të bëhet program ku do të krijohet funksioni me emrin "funksion_pa_parametra" i tipit **void**, ku brenda tij të shtypet teksti *"Funksion i tipit void pa parametra."*. Thirrja e funksionit të bëhet nga programi **main**.

```
# include <iostream>
using namespace std;

void funksion_pa_parametra()                    // deklarimi i funksionit pa parametra
{                                               //fillimi i trupit të funksionit
   cout << "\n\tFunksion i tipit void pa parametra." << endl;
}                                               //mbarimi i trupit të funksionit
```

```
int main()
{
    funksioni();                    //thirrja e funksionit brenda trupit main

   return 0;
}
```

Sqarim:

Tek ky program vërejmë se nuk kemi prototipin e funksionit. Kjo për arsye se funksioni në fjalë është shënuar para funksionit kryesorë, e në këtë rast nuk shënohet prototipi. Por nëse funksionin e dhënë e shënojmë pas funksionit kryesorë atëherë deklarimi i prototipit është i patjetërsueshëm. Deklarimi i prototipit bëhet në këtë mënyrë:

```
void funksion_pa_parametra()  ;
```

Pas ekzekutimit të programit rezultati në ekran do të jetë teksti:

```
Funksioni i tipit void pa parametra.
```

Në këtë rast mund të parashtrohet pyetja:

"Si bëhet ekzekutimi i programit kur kemi funksion?"

Shënim:

Ekzekutimi i programit në këtë rast bëhet në të njëjtën mënyrë siç bëhet zakonisht edhe te programet e deritanishme që kemi mësuar. Pra, ekzekutimi bëhet resht për reshti (komand për komande) duke filluar nga fillimi i programit. Mirëpo kur kompajleri të ketë arritur tek komanda përmes së cilës e thërrasim funksionin atëherë automatikisht kërcen tek fillimi i funksionit i cili thirret ku do qoftë ai në program (para apo pas programit kryesor) dhe fillon ekzekutimin e atij funksioni. Dhe po që se ka ende komanda apo funksione pas funksionit në fjalë kompajleri nuk vazhdon më poshtë por kthen përsëri mbrapa tek komanda për thirrjen e funksionit kur të ketë mbaruar me ekzekutimin e funksionit të thirrur dhe vazhdon që aty përsëri. Këtë gjë e kemi sqaruar më mirë përmes një shembulli në vazhdim.

Shembull3: Të bëhet program ku do të krijohet funksioni me emrin "funksion_pa_parametra" i tipit **void**, ku brenda tij të shtypet teksti *"Funksion i tipit void pa parametra."* Thirrja e funksionit të bëhet nga programi **main**. Pas thirrjes së funksionit të nxirret në ekran teksti *"Funksioni i tipit void me parametra."*

```
# include <iostream>
using namespace std;

void funksion_pa_parametra()                    // deklarimi i funksionit pa parametra
{                                               //fillimi i trupit të funksionit
   cout << "\n\tFunksion i tipit void pa parametra." << endl;
}                                               //mbarimi i trupit të funksionit
int main()
{
      funksioni();                              //thirrja e funksionit brenda trupit main
     cout << "Funksioni i tipit void me parametra." << endl;
   return 0;
}
```

Pas ekzekutimit të programit në Code::Blocks dalja në ekran do të jetë:

```
Funksioni i tipit void pa parametra.
Funksioni i tipit void me parametra.
```

Në vazhdim do të marrim një shembull të thjeshtë ku në të do të deklarojmë një funksion të tipit void (pa parametra) por që do e shënojmë pas funksionit kryesorë.

Shembull4: Të bëhet program ku do të krijohet funksioni me emrin "shuma" i tipit **void**, ku brenda tij të llogaritet formula: (5x+2y)/2, ku x = 2 dhe y = 5. Thirrja e funksionit të bëhet nga programi **main**.

```
# include <iostream>
using namespace std;

void shuma();                    // deklarimi i prototipit të funksionit
int main()
{
      cout << "Shuma: " << funksioni();          //thirrja e funksionit brenda trupit main
      return 0;
}
void shuma()                     // deklarimi i funksionit pa parametra
{                                                //fillimi i trupit të funksionit
   int x = 2, y = 5;
   double z;
   z = (5*x+2*y)/2;
   cout << z <<endl;
}                                                //mbarimi i trupit të funksionit
```

Shënim:

Pra siç shohim te programi në fillim është shënuar prototipi i funksionit pastaj programi kryesorë dhe më pas funksioni në fjalë. Njëjtë veprohet edhe po të kemi më shumë funksione pas funksionit kryesorë, pra duhet të shënohet prototipi në fillim. Nëse funksioni shënohet para funksionit kryesorë prototipi nuk shënohet.

Funksione pa vlerë kthyese me parametra

Këto funksione përmbajnë parametra, pra nga vendi ku thirren marrin parametra (vlera). Parametrat shënohen brenda kllapave (), dhe në mes veti të ndarë me presje, për shembull:

```
void funksioni (int parametri1, int parametri2);
```

Kurse thirrja e funksionit në programin kryesorë bëhet duke e shënuar emrin e funksionit, kllapat matematikore () dhe brenda kllapave parametrat. Thirrja do të duket kështu:

```
funksioni (parametri1, parametri2);
```

Gjithashtu tipi i parametrit në radhë tek thirrja e funksionit duhet të korrespondojë me tipin e parametrit në radhë tek deklarimi i funksionit.

Për shembull:

Nëse kemi të deklaruar këtë funksion:

```
void funksioni (int parametri1, double parametri2);
```

Thirrja e tij duhet të jetë kështu:

```
funksioni (parametri1, parametri2);
```

pra tipi i parametrit në radhë tek thirrësi duhet të jetë identik me atë tek deklarimi i funksionit, këtu parametri2 duhet të jetë i tipit **double**.

Gjë tjetër e rëndësishme është emri i parametrit i cili mund të jetë i ç'far deshëm, pra emri i parametrit tek thirrësi nuk ka nevojë të jetë identik me emrin e parametrit tek funksioni. Kjo do të thotë se ato mund të jenë të ndryshëm, shembull:

```
void funksioni (int parametri1, int parametri2);  // deklarimi i funksionit
```

```
funksioni (vlera1, vlera2);          // thirrja e funksionit
```

Po ashtu edhe numri i parametrave tek thirrësi i funksionit duhet të jetë identik me numrin e parametrave tek funksioni.
Në vazhdim kemi marrë një shembull i cili do të llogaris shumën e *n* numrave natyrorë.

Shembull4: Të bëhet program ku do të krijohet funksioni me emrin "shumaN" i tipit **void**, ku brenda tij të llogaritet shuma e numra natyrorë nga 1 deri në n. Vlera e n të futet nga tastiera dhe ti jepet funksionit gjatë thirrjes së tij. Thirrja e funksionit të bëhet nga programi **main**, kurse rezultati të shfaqet në ekran.

```
# include <iostream>
using namespace std;

void shumaN(int a);        // deklarimi i prototipit të funksionit me parametër
int main()
{
      int n;
      cout << "Jep n: "
      cin >> n;
      cout << "Shuma: " << funksioni(n);   //thirrja e funksionit brenda trupit main
      return 0;
}
void shumaN(int a)                  // deklarimi i funksionit me parametër
{                                   //fillimi i trupit të funksionit
   int shuma = 0;
   for (int i = 1; i <= a; i++)
   {
         Shuma += i;
   }
   cout << shuma << endl;
}                                   //mbarimi i trupit të funksionit
```

Le të analizojmë programin e mësipërm dhe do të kemi:

Së pari: është bërë deklarimi i prototipit të funksionit i cili është shënuar pas funksionit kryesorë (main). Ky deklarim duket si në vijim:

void shumaN(int a);

Së dyti: në programin kryesorë kemi deklaruar një variabël të tipit **int** ku përmes këtij variabëli do të fusim n numrat natyrorë që do të mblidhen. Përmes komandës *cout* dhe *cin* i lejojmë shfrytëzuesit që të fus këtë numër. Pastaj nxjerrim rezultatin në ekran përmes rreshtit:

cout << "Shuma: " << funksioni(n);

ku në ekran do të nxirret teksti Shuma: dhe pastaj thirret shuma e n numrave natyrorë. Pra, thirret funksioni *shumaN* ku llogarit shumën e n numrave. Kjo realizohet përmes thirrjes së funksionit:

funksioni(n);

ku i jepet edhe vlera **n**.

Së treti: funksioni fillon me llogaritjen e **n** numrave të parë natyrorë, ku vlera e **n**-së i shoqërohet variabëlit **a**, pastaj kemi deklaruar variabëlin shuma me vlerë fillestare 0 si vijon:

```
int shuma = 0;
```

kjo për arsye se llogaritet në cikël.
Dhe përmes ciklit:

```
for (int i = 1; i <= a; i++)
{
        Shuma += i;
}
```

llogaritet shuma e n numrave natyrorë dhe me komandën:

```
cout << shuma << endl;
```

nxjerrim në ekran shumën e n numrave natyrorë.
Vërejtje: Funksioni shumaN është funksion me parametra por që nuk kthen vlerë dhe do të ishte e kotë po që se nuk e shënonim reshtin:

```
cout << shuma << endl;
```

tek funksioni shumaN, pasi në ekran nuk do të nxirreshe asgjë, por vetëm do tu llogaritkëshe shuma e *n* numrave natyrorë.

Funksionet me vlerë kthyese

Ky lloj i funksioneve është më se i përdorur dhe i domosdoshëm po thuajse në të gjithë programet. Këto funksione kthejnë vlerë tek thirrësi i tij, dhe më e rëndësishmja se këto vlera mund më tej të përdoren si variabëla. Edhe ky lloj funksioni ndahet në:

- Funksione me vlerë kthyese pa parametra, dhe
- Funksione me vlerë kthyese me parametra.

Funksione me vlerë kthyese pa parametra

Funksionet me vlerë kthyese pa parametra nuk përmbajnë parametra, pra nga vendi ku thirren (zakonisht nga programi kryesorë) nuk marrin parametra (vlera).
Në vazhdim do të kemi një shembull të thjeshtë ku do të llogarisim syprinën e rrethit.

Shembull5: Të bëhet program ku do të krijohet funksioni me emrin "funksion_pa_parametra" i tipit **non-void** (int, që kthen vlerë), ku brenda tij të llogaritet syprina e rrethit. Thirrja e funksionit të bëhet nga programi **main**, ku rezja të jetë 10 dhe funksioni të kthen syprinën e rrethit. Pastaj të nxirret në ekran syprina.

```
# include <iostream>
using namespace std;
int funksion_pa_parametra()                    // deklarimi i funksionit pa parametra
{
  const int pi = 3.1415;                       //fillimi i trupit të funksionit
  int rrezja = 10;
  int Syprina;
  Syprina = rrezja * rrezja * pi;
  return Syprina;
}                                              //mbarimi i trupit të funksionit
int main()
{
      int S;
      S = funksioni();                         //thirrja e funksionit brenda trupit main
      cout << "Syprina e rrethit = " << S;
   return 0;
}
```

Shënim

Siç shihet në program funksioni në fjalë kthen vlerë tek thirrësi i tij, prandaj thirrja pra vlera që kthehet patjetër ti shoqërohet ndonjë variable, në rastin tonë variabëlit S:

```
S = funksioni();
```

dhe pastaj nxirret në ekran.

Ky funksion është funksion që kthen vlerë tek thirrësi i tij prandaj është i tipit **non-void**, pra që do të thotë *jo-i pavlefshëm*. Prandaj në këtë rast para tij nuk shënohet tipi **void** por ndonjë

tip tjetër në rastin tonë **int**. Çka do të thotë kjo? – kjo do të thotë se funksioni në fjalë vlerën kthyese që do ta kthejë do të jetë e tipi integer (numër i plotë), edhe në qoftë se brenda funksionit kemi llogaritje me presje dhjetore. Në këtë rast kur kemi llogaritje me numra dhjetorë funksionin e deklarojmë të tipit float ose double, si për shembull:

double funksion_pa_parametra()

gjithashtu funksionet e tipit **non-void** (që kthejnë vlerë) mund të deklarohen edhe të tipave të ndryshëm siç janë: **short, char, string, short int** etj.

Funksione me vlerë kthyese me parametra

Ky lloj funksioni është funksion që merr dhe kthen vlerë nga thirrësi i tij. Edhe nga përshkrimi i tij duket se është më i avancuar dhe pa dyshim më i përdorur tek programet.
Në vazhdim kemi një shembull ku do të ilustrojmë këtë lloj funksioni.

Shembull6: Të bëhet program ku do të krijohet funksioni me emrin "funksion_me_parametra" i tipit **non-void** (**int**, që kthen vlerë), ku brenda tij të llogaritet syprina e rrethit. Thirrja e funksionit të bëhet nga programi **main**, ku vlera e rrezes e futur nga tastiera ti jepet funksionit, i cili kthen syprinën e rrethit. Pastaj të nxirret në ekran syprina.

```
# include <iostream>
using namespace std;

int funksion_me_parametra(int rrezja)          // deklarimi i funksionit me parametra
{
  const int pi = 3.1415;                       //fillimi i trupit të funksionit
  int Syprina;
  Syprina = rrezja * rrezja * pi;
  return Syprina;
}                                              //mbarimi i trupit të funksionit
int main()
{
      int S, r;
      cout << "Vlera e rrezes: ";
      cin >> r;
      S = funksioni(r);                        //thirrja e funksionit brenda trupit main
      cout << "Syprina e rrethit = " << S;
   return 0;
}
```

Siç shohim në program funksioni përmban parametra dhe atë:

int rrezja

ku kjo variabël merr vlerën e variabëlës që shënohet brenda kllapave tek thirrësi i funksionit (ose komanda për thirrjen e funksionit):

```
S = funksioni(r);
```

dallimi ndonëse është i vogël por programi në fjalë është më i avancuar se sa ai tek shembulli5. Kjo për arsye se ky program mund të përdoret për ç'far do vlere të futur të rrezes r dhe do të kemi rezultat të ndryshëm sa herë që ndryshojmë vlerën e rezes, kurse tek programi në shembullin5 kemi një zgjidhje dhe atë për vlerën e rrezes që është 10.

Përveç kësaj ndarje bazë të funksioneve, kemi edhe lloje tjera si për shembull: funksione inline, rekursive etj., që do ti sqarojmë më vonë.

Shtrirja Globale dhe Lokale

Çdo gjë që është definuar (deklaruar) në nivel të shtrirjes së programit (p.sh. jashtë funksioneve dhe klasave) është e thënë të ketë një shtrirje globale. Kështu që funksionet e thjeshta që kemi përdorur deri tani të gjithë kanë shtrirje globale. Variablat gjithashtu mund të definohen në shtrirje globale:

```
int year = 1994; // global variable
int Max (int, int); // global function
int main (void) // global function
{
        //...
}
```

Variablat globale të pa inicializuara, automatikisht janë të inicializuara me vlerën zero.

Që kur entitetet globale janë të dukshme në nivelin programit, ata gjithashtu duhet të jenë unike (të vetme) në atë nivel të programit. Kjo do të thotë se variabla apo funksione të njëjta nuk mund të definohen më shumë se një herë në nivel global. Entitetet globale janë përgjithësisht të qasshme kudo në program.

Çdo bllok[1] në program përcakton një *shtrirje (fushë) lokale*. Pra trupi i një funksioni prezanton një shtrirje lokale. Parametrat e funksionit kanë shtrirje të njëjtë si trupi i funksionit. Variablat e definuar në fushën lokale janë të qasshme vetëm në atë fushë. Kështu që një ndryshore

[1] Një bllok në program do të thotë nga çasti i hapjes së kllapave gjarpëruese {, dhe deri te mbyllja e tyre }. Për shembull një funksion do të thotë një bllok, një cikël *for* një bllok tjetër apo një cikël *if* ose *while, do while, switch* etj., të gjithë këto paraqesin blloqe.

(variabël) duhet të jetë unike (e vetme) vetëm brenda fushës (shtrirjes) së vetë. Fushat lokale mund të jenë të mbivendosura, në të cilin rast fushat e brendshme i refuzojnë fushat e jashtme. Për shembull:

```
int xyz;                    // xyz është globale
void Foo (int xyz)          // xyz është lokale në trupin e funksionit  Foo
{
if (xyz > 0)
{
        double xyz;         // xyz është lokale në këtë bllok
        //...
    }
}
```

ka tre fusha të ndryshme, dhe secila fushë përmban nga një ndryshore xyz të ndryshme.

Në përgjithësi jetëgjatësia e një ndryshore është e kufizuar brenda fushës ku ajo përfshihet. Kështu, për shembull, ndryshoret globale zgjat me kohëzgjatjen e ekzekutimit të programit, kurse ndryshoret lokale janë krijuar kur fusha e tyre ka filluar dhe shkatërrohen (fshihen) kur fusha e tyre të ketë përfunduar. Hapësira e memories për ndryshoret globale është e rezervuar para se të fillon ekzekutimi i programit, kurse hapësira e memories për ndryshoret lokale është e ndarë në fly për gjatë ekzekutimit të programit.

Operatori i fushës (shtrirjes)

Nga që fusha lokale e refuzon fushën globale, një variabël lokale me emër të njëjtë si një variabël globale e bën këtë të fundit të pa qasshme në fushën lokale. Për shembull në

```
int error;
void Error (int error)
{
        //...
}
```

ndryshorja globale error është e pa qasshme brenda funksionit Error, sepse është e kufizuar nga ndryshorja lokale error. Ky problem është i

kapërcyer duke përdorur operatorin unar të fushës (shtrirjes, hapësirës) :: që merr një entitet global si argument.

```
int error;
void Error (int error)
{
        //...
if (::error != 0)          // refers to global error
        //...
}
```

tani më përmes operatorit :: (katër pika) mund ti qasemi ndryshores globale error.

Ndryshoret auto (automatike)

Për shkak se jetëgjatësia e ndryshoreve lokale është e kufizuar dhe e vendosur automatikisht, këto ndryshore gjithashtu janë quajtur edhe automatike. Klasa ruajtëse e specifikuar si auto mund të përdoret që qartë të specifikoj variabëlin lokale të bëhet automatike.

```
void Foo (void)
{
auto int xyz;              // ngjashëm si: int xyz;
//...
}
```

Kjo është përdorur shumë rrallë, sepse të gjitha ndryshoret lokale kompjuteri i njeh si automatike.

Ndryshoret regjistruese

Siç është përmendur më herët, ndryshoret zakonisht paraqesin lokacione të memories ku ruhen vlerat e ndryshores. Kur kodi programor i referohet një ndryshoreje (p.sh., në një shprehje), kompajleri gjeneron kodin e makinës që bën të qasshëm lokacionin e memories të paraqitur nga ndryshorja. Për ndryshoret që përdoren më shpesh (p.sh., ndryshoret

ciklike[2]), fitimet efektive mund të sigurohen duke mbajtur ndryshoren në regjistër kështu i shmangemi qasjes në memorie për atë ndryshore.

Klasa ruajtëse e cilësuar *regjistër* mund të përdoret për ti treguar kompajlerit se ndryshorja do të ruhet në regjistër nëse është e mundur. Për shembull:

```
for (register int i = 0; i < n; ++i)
sum += i;
```

Tani këtu, në çdo cikël, *i*-ja përdoret tre herë: një kur krahasohet me *n*, së dyti kur mblidhet me *sum* dhe së treti kur ritet. Për këtë arsye përdorim ndryshoret *regjistër* që do ta mbajnë atë ndryshore në regjistër për aq kohë sa zgjatet cikli.

Njoftohet se regjistri është vetëm një shenjë paralajmëruese (informatë) tek kompajleri, dhe në disa raste kompajleri mund të zgjedh të mos e përdor regjistrin kur ai pyetet për ta bërë pra. Një arsye për këtë është ajo se ndonjë makinë (kompjuter) ka numër të limituar të regjistrave dhe kjo mund të jetë një shkak që ata të gjithë janë në përdorim.

Njëjtë kur programori nuk e përdor deklarime të regjistrit, shumë kompajler optimist provojnë të marrin një supozim inteligjent dhe përdorin regjistrat kur ata janë të mundshëm të risin performancat e programit.

Përdorimi i deklarimeve të regjistrit mund të lihen si një mendim i mëvonshëm; ata gjithashtu mund të vendohen më vonë me rishikim të kodit dhe vendosjen e tyre në vende të përshtatshme.

Ndryshoret dhe funksionet statike

Gjithashtu është e dobishme që të kufizohet mundësia e qasjes tek ndryshoret globale ose funksionet në një skedar (fajll) të vetëm. Kjo është e mundshme nga klasa ruajtëse e cilësuar **static**. Për shembull, konsiderojmë një program i lojës mister (puzzle) që përbëhet nga tre fajlla, për gjeneratat e lojës, zgjedhjet e lojës, dhe ndërfaqja me shfrytëzuesin (user interface). Fajlli i zgjedhjeve të lojës, duhet të përmbajë funksionin *Solve* dhe numrin e funksioneve tjera ndihmëse të funksionit *Solve*. Nga që këta të fundit janë vetëm për përdorim privat të

[2] Ciklike – fjala origjinale në anglisht është fjala loop, që ka kuptimin lak, por për të përshtatur dhe kuptuar më mirë themi ndryshore ciklike që përdoret në cikle. Për shembull ndryshorja e ciklit for, while, do while etj.

funksionit *Solve*, është më mirë që ti bëjmë ato mos të kenë qasje jashtë fajllit:

```
static int FindNextRoute (void)     // i qasshëm vetëm në këtë faljll
{
        //...
}
        //...
int Solve (void)                    // i qasshëm edhe jashtë fajllit
{
        //...
}
```

Argumenti i njëjtë mund të përdoret edhe për ndryshoret globale në këtë fajll që janë për përdorim privat të funksioneve në fajll. Për shembull, një ndryshore globale që përcakton gjatësinë e rrugës më të shkurtër deri më tani është mirë e definuar si statik.

```
static int shortestRoute;           // ndryshorja globale statike
```

Ndryshorja lokale brenda funksionit gjithashtu mund të definohet statike. Ndryshorja do të mbetet e arritshme (qasshme) vetëm brenda fushës së sajë lokale; megjithatë, jetëgjatësia e sajë nuk do të jetë e mbyllur në këtë fushë, por përkundrazi do të jetë globale. Me fjalë të tjera, një variabël lokale statike është një variabël globale e cila është e arritshme vetëm fushëveprimit të sajë lokal. Ndryshoret statike lokale janë të dobishme kur ne duam që vlera e ndryshores lokale të vazhdoj për të gjithë thirrjet për funksionin në të cilin ajo shfaqet. Për shembull, e konsiderojmë një funksion *Error* që mban numërimin e gabimeve dhe ndërpret programin kur numërimi të ketë arritur limitin paraprakisht të shënuar:

```
void Error (char *message)
{
static int count = 0;       // ndryshorja lokale statike
if (++count > limit)
Abort();
//...
}
```

Sikurse ndryshoret globale, edhe ndryshoret lokale statike janë automatikisht të inicializuara[3] 0.

[3] Automatikisht kanë vlerë fillestare 0.

Ndryshoret dhe funksionet eksterne (të jashtme)

Nga se një ndryshore globale mund të definohet në një fajll dhe referuar në fajlla tjerë, mënyra për ti treguar kompajlerit se ndryshorja është definuar tjetërkund është e nevojshme. Përndryshe, kompajleri mund ta kuptoj ndryshoren si të pa definuar. Kjo është e lehtësuar nga deklarimi i jashtëm (extern).

```
extern int size;          // deklarim i ndryshores
```

e informon kompajlerin se ndryshorja *size* aktualisht është e definuar diku (mund të jetë më vonë në këtë fajll ose në tjetër). Kjo thirret deklarim i ndryshores (jo definim) sepse ajo nuk të çon te ndonjë ruajtje e caktuar ekzistuese për ndryshoren *size*.

Është një stërvitje e dobët programuese për të përfshirë një inicializim për një ndryshore të jashtme (extern), meqenëse kjo e bën atë të jetë ndryshore e definuar dhe ka ruajtje të caktuar për të:

```
extern int size = 10;     // jo më deklarim
```

Nëse këtu ka një përcaktim (definim) tjetër për ndryshoren *size* ndokund në program, ajo përfundimisht do të ndeshet me këtë.

Prototipat e funksioneve gjithashtu mund të deklarohen si të jashtëm (extern), por kjo nuk ka efekt kur prototipi gjendet në fushën globale (shtrirjen globale). Është më e dobishme për deklarim të prototipave të funksioneve brenda funksionit. Për shembull:

```
double Tangent (double angle)
{
        extern double sin(double);      // definuar tjetërkund
        extern double cos(double);      // definuar tjetërkund
        return sin(angle) / cos(angle);
}
```

Vendi më i mirë për deklarimet eksterne është zakonisht në krye fajllat (header files) kështu që ato mund lehtë të përfshihen dhe të shpërndahen nga fajlli burimor.

Konstantet simbolike

Duke ia shtuar përpara fjalën *const* dfinimit të ndryshores e bëjmë atë ndryshore vetëm të lexueshme (read-only, p.sh. një konstante simbolike). Konstantja patjetër të inicializohet me ndonjë vlerë kur të definohet. Për shembull:

```
const int maxSize = 128;
const double pi = 3.141592654;
```

Pasi njëherë të definohet, vlera e konstantes nuk mund të ndryshohet:

```
maxSize = 256;          // e ndaluar!
```

Konstantja pa tip të caktuar është e supozuar të jetë *int*:

```
const maxSize = 128;    // maxSize është e tipit int
```

Me printerët, duhet të konsiderohen dy aspekte: vetë treguesi (pointeri), dhe objekti i treguar (pointuar). Cili do nga këto ose të dy bashkë mund të jenë konstant.:

```
const char *str1 = "pointer to constant";
char *const str2 = "constant pointer";
const char *const str3 = "constant pointer to constant";
str1[0] = 'P';                  // ndaluar!
str1 = "ptr to const";          // rregull
str2 = "const ptr";             // ndalur!
str2[0] = 'P';                  // rregull
str3 = "const to const ptr";    // ndluar!
str3[0] = 'C';                  // ndaluar!
```

Edhe parametrat e funksionit mund të deklarohen konstant. Kjo mund të përdoret për të treguar se funksioni nuk e ndryshon vlerën e parametrave:

```
int Power (const int base, const unsigned int exponent)
{
        //...
}
```

Funksioni gjithashtu mund të kthejë rezultat konstant:

```
const char* SystemVersion (void)
{
return "5.2.1";
}
```

Vendi i zakonshëm për përcaktimin e konstanteve është brenda krye fajllave (header files), kështu që ato mund të shpërndahen nga fajlli burimor.

Numëruesit

Një numërues i konstanteve simbolike është e paraqitur me një deklarim *enum*. Kjo është e dobishme për deklarimin e konstanteve të lidhura ngushtë njëra me tjetrën. Për shembull:

```
enum {veri, jug, lindje, perëndim};
```

paraqet katër numërues që kanë numër të plotë që fillon prej 0 (p.sh., veri është 0, jug është 1, lindje 2, perëndim 3). Ndryshëm nga konstantet simbolike, megjithatë, që janë ndryshore vetëm që lexohen (read-only), numëruesit nuk kanë memorie të ndarë.

Mosplotësimi i numërimit të numëruesve mund të refuzohet me inicializim të saktë:

```
enum {veri= 10, south, lindje = 0, perëndim};
```

këtu *jug* është 11 dhe *perëndim* 1.

Një numërues gjithashtu mund të emërohet, ku emri bëhet tip i përcaktuar nga shfrytëzuesi. Kjo është e dobishme për definimin e ndryshoreve të cilat vetëm mund të bëhen të caktuara për një numër të limituar të vlerave. Për shembull, në

```
enum Drejtimi{veri, jug, lindje, perëndim};
Drejtimi d;
```

d vetëm mund të përcaktoj një nga numëruesit për *Drejtimi*.

Numëruesit janë posaçërisht të dobishëm për emërtimin e rasteve të degëzimit *switch* .

```
switch (d) {
case veri:      //...
case jug:       //...
case lindje:    //...
case perëndim:  //...
}
```

Ne në mënyrë të gjerë do t përdorim numëruesin për paraqitjen e vlerave logjike (boolean) në program:

```
enum Bool {false, true};
```

?Runtime Stack

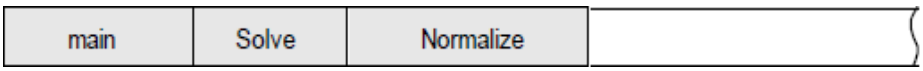
Ngjajshëm si shumë gjuhë tjera programuese moderne, thirrja e ekzekutimit të C++ funksioneve është e bazuar në runtime stack. Kur funksioni thirret, vendi i memories është i vendosur në këtë stack për parametrat e funksionit, vlerën kthyese, dhe ndryshoret lokale, më mirë se sa hapësira lokale e stack për vlerësimin e shprehjes. Vendi i ndarë thirret stack frame. Kur funksioni kthen (vlerë), stack frame i ndarë është i çliruar, kështu që ai mund të ripërdoret.

Për shembull, konsiderojmë situatën ku funksioni kryesorë main thirr funksionin e quajtur Solve i cili në kthim thërret një funksion tjetër të quajtur Normalize:

```
int Normalize (void)
{
        //...
}
int Solve (void)
{
        //...
Normalize();
        //...
}
int main (void)
{
        //...
Solve();
        //...
}
```

Figura 4.5 ilustron stack frame kur ekzekutohet funksioni Normalize.

Stack Frames të funksionit të thirrur.



Është e rëndësishme të thuhet se thirrja e funksionit përfshin varshmërinë (overheads) e krijimit të stack frames për vete dhe fshirjen e

tyre kur kthen vlerë. Për shumë funksione kjo varshmëri është e pa përfillshme krahasuar me kalkulimin aktual që funksioni përmbush.

Funksionet Inline

Supozojmë se programi herë pas here kërkon të gjejë vlerën absolute të një madhësie të plotë (integer). Për vlerën e paraqitur me *n*, kjo mund të pasqyrohet si:

```
(n > 0 ? n : -n)
```

Megjithatë, në vend që të zhvendosim këtë shprehje në shumë vende në program, është më mirë që ta definojmë atë si funksion:

```
int Abs (int n)
{
return n > 0 ? n : -n;
}
```

Versioni i funksionit ka një numër të përparësive. Së pari, kjo të çon në më shumë programe të qarta (kuptueshme). Së dyti, kjo është e ripërdorshme. Dhe së treti, shmang efektet anësore të padëshirueshme kur argumenti është vet një shprehje me efekt anësorë.

Mangësi e versionit të funksionit, prapëseprapë, është ajo se frekuenca e përdorimit të sajë shpie në ndëshkim të konsiderueshëm të performancave, pikërisht në varshmërinë lidhur me thirrjen e funksionit. Për shembull, nëse përdoret funksioni *Abs* pa cikël që përsëritet qindra herë, atëherë ai do të ketë një ndikim (përplasje) në performancë. Varshmëria mund të mënjanohet duke e definuar funksionin *Abs* si një funksion **inline**:

```
inline int Abs (int n)
{
        return n > 0 ? n : -n;
}
```

Efekti i kësaj është se kur thirret funksioni *Abs*, kompajleri, në vend që të gjeneroj kodin për thirrjen e funksionit *Abs*, zgjeron dhe zëvendëson trupin e funksionit *Abs* në vend të thirrjes. Ndërsa para së gjithash llogaritja e njëjtë është kryer, as një thirrje e funksionit nuk është përfshirë, kështu që as një stack frame nuk është caktuar.

Pasi që thirrjet në një funksion janë të zgjeruara, as një gjurmë e funksionit nuk do të mbetet në kodin e kompiluar. Prandaj, nëse një funksion në një fajll është i definuar si inline, ai nuk do të jetë i mundshëm në fajllat tjerë. Si pasojë, funksionet inline zakonisht është i vendosur në krye fajllat kështu ato mund të shpërndahen. Ashtu siç fjala kyçe *register*, *inline* është një shenjë që kompajleri nuk është i obliguar ta zbatojë. Përgjithësisht, përdorimi i fjalës kyçe inline do të jetë e kufizuar në mënyrë të qartë, herë pas here gjatë përdorimit të funksioneve. Përdorimi i inline për funksionet e tej gjata dhe komplekse është po thuajse e sigurt e pranuar nga kompajleri.

Rekursivi

Funksioni që e thirr veten e tij është thënë të jetë rekursiv. Rekursivi është një teknikë e përgjithshme programimi e zbatueshme në probleme të cilët mund të definohen në marrëdhënie me veten. Marrim problemin e faktorielit, për rastin, që është i definuar si:

- Fakroieli i 0 është 1.
- Fakroieli i numrit pozitiv *n* është *n* – herë faktorieli i n-1.

Reshti i dytë qartë tregon se faktorieli është definuar në marrëdhënie me vetveten dhe që këtu mund të pasqyrohet si funksion rekursiv:

```
int Factorial (unsigned int n)
{
        return n == 0 ? 1 : n * Factorial(n-1);
}
```

Për *n* e barabartë me 3, Tabela në vazhdim jep gjurmët e thirrjes së funksionit *Factorial*. Stack frames për këto thirrje shfaqin vazhdueshmërinë runtime stack, njëra pas tjetrës.

Thirrja	n	n == 0	n * Factorial (n-1)	Kthimi
e parë	3	false	3 * Factorial (2)	6
e dyta	2	fasle	2 * Factorial (1)	2
e treta	1	fasle	1 * Factorial (0)	1
e katërta	0	frue		

Funksioni rekursiv patjetër të ketë më pak se një kusht të fundit që do të jetë i plotësuar. Ndryshe, funksioni do ta thërret vetveten pacaktueshëm deri sa runtime stack do të mbushet. Funksioni *Factorial*, për shembull, e ka kushtin e fundit *n == 0* që, kur plotësohet, shkakton thirrjet rekursive të kthen mbrapa prapa. (Vëmendje për atë se për *n* negative ky kusht nuk do të plotësohet dhe *Factorial* do të dështoj).

Si rregull e përgjithshme, të gjithë funksionet rekursive mund të rishkruhen duke përdorur përsëritjen. Në situata kur numri i stack frames të përfshira mund të jetë lirshëm i gjerë, versioni i përsëritur është i nevojshëm. Në raste tjera, eleganca dhe thjeshtësia e versionit rekursiv mund ti japë asaj përparësi.

Për faktorielin, për shembull, një argument shumë i madh, mund të çojë në shumë stack frames. Një version i përsëritjes është i nevojshëm në këtë rast:

```
int Factorial (unsigned int n)
{
        int result = 1;
        while (n > 0) result *= n--;
        return result;
}
```

Argumentet fillestare (default[4])

Argumenti fillestar (argument i plotësuar), është një lehtësim i programimit që mënjanon ngarkesën për të specifikuar vlerat e argumenteve për të gjithë parametrat e funksionit. Për shembull, konsiderojmë funksionin për raportimin e gabimeve:

```
void Error (char *message, int severity = 0);
```

Këtu, severity ka vlerë fillestare 0; prandaj të dy thirrjet në vazhdim janë të vlefshme:

```
Error("Division by zero", 3);      // severity e caktuar në 3
```

[4] Default – nga fjala angleze që në përkthim do me thënë mosplotësim, mirëpo në këtë rast ka kuptimin për argumentet të cilat janë të inicializuar me vlera prej momentit të deklarimit të tyre, prandaj më mirë do të ishte që në vend të kësaj fjale të përdorim fjalën fillestare (p.sh., vlera fillestare e një ndryshoreje n është e barabartë me zero, *n=0*).

Error("Round off error"); *// **severity** e caktuar në 0*

Siç ilustron thirrja e parë, vlera fillestare mund të refuzohet duke e cilësuar qartë një argument.

Argumentet fillestare janë të përshtatshme për situata kur disa (ose të gjitha) parametra të funksionit herë pas here marrin vlerat e njëjta. Në funksionin *Error*, për shembull, rreptësisht *0* gabime janë më shumë unike se tjerat dhe atypari kandidat i mirë për vlerë fillestare. Pa përdorimin e duhur të argumenteve fillestare do të ishte:

int Power (int base, unsigned int exponent = 1);

Nga që *1* (ose ndonjë vlerë tjetër) është pa përshtatshëm të përdoret herë pas here në këtë situatë.

Për të shmangur dykuptimësinë, të gjithë argumentet fillestare patjetër të jenë argumente zvarritëse. Deklarimi në vazhdim prandaj është ilegal:

void Error (char *message = "Bomb", int severity); ***// I ndaluar!***

Argumenti fillestar nuk duhet të jetë domosdoshëm konstant. Shprehjet atribute mund të përdoren, përgjatë sa ndryshoret përdoren në shprehje janë në dispozicion në fushë (hapësirë) të funksionit të definuar. (p.sh., ndryshoret globale).

Marrëveshja e pranuar për argumentet fillestare është të specifikoj ato në deklarimet e funksionit, jo në përcaktimet e funksionit. Nga se deklarimet e funksionit shfaqen në krye fajllat, kjo mundëson shfrytëzuesin e funksionit të ketë kontroll mbi argumentet fillestare. Pra argumentet fillestare të ndryshme mund të specifikohen për situata të ndryshme. Kjo është, prapëseprapë, e ndaluar të specifikomë dy argumente fillestare të ndryshme për të njëjtin funksion në një fajll.

Numri i ndryshoreve të argumenteve

Ndonjëherë është e përshtatshme, nëse jo e domosdoshme, të kemi funksione që marrin një varg ndryshoresh të argumenteve. Një shembull i thjeshtë është funksioni i cili merr disa alternativa të menysë si argumente, shfaq menynë, dhe lejon shfrytëzuesin që të zgjedh një nga alternativat. Të jetë i përgjithshëm, funksioni duhet të jetë në gjendje të

pranoj ndonjë numër të alternativave si argumente. Kjo mund të pasqyrohet si:

int Menu (char *option1 ...);

që konstaton se *Menu* mund të marrë një apo më shumë argumente.

Menu mund t'u qaset argumenteve të veta duke përdorur disa përcaktime makro në krye fajllin **stdarg.h**, si është ilustruar në Shembullin 7. Makrot e përshtatshëm janë nënvizuar në boald.

Shembull7:

```
1     #include <iostream.h>
2     #include <stdarg.h>
3     int Menu (char *option1 ...)
4     {
5     va_list args;                          // argumentojmë list
6     char* option = option1;
7     int count = 0, choice = 0;
8     va_start(args, option1);               // e inicializojm args
9     do {
10    cout << ++count << ". " << option << '\n';
11    } while ((option = va_arg(args, char*)) != 0);
12    va_end(args);                          // e pastrojmë args
13    cout << "option? ";
14    cin >> choice;
15    return (choice > 0 && choice <= count) ? choice : 0;
16    }
```

Shënim:

5 Për t'iu qasur argumenteve, ***args*** është deklaruar të jetë e tipit *va_list.*

8 ***Args*** është e inicializuar duke thirrur *va_start*. Argumenti i dytë tek *va_start* patjetër të jetë parametri i fundit i funksionit saktësisht i deklaruar në krye funksionin (p.sh., në këtë rast *option1*).

11 Argumentet pasues janë gjetur duke thirrur *va_arg*. Argumenti i dytë tek *va_arg* patjetër të jetë tip i pritur nga ai argument (p.sh., në këtë rast *char**). Kjo teknikë që të punojë, argumenti i fundit patjetër të jetë 0, duke shënuar fundin e argument listës. *Va_arg* është thirrur vazhdimisht përderisa është arritur 0.

12 Më në fund, është thirur *va_end* që të kthej runtime stack (që ndoshta ka qenë modifikuar nga thirrjet e më hershme).

Thirrja e thjeshtë

```
int n = Menu(
        "Open file",
        "Close file",
        "Revert to saved file",
        "Delete file",
        "Quit application",
        0);
```

Ku dalja në ekran pas ekzekutimit do të jetë:

```
1. Open file
2. Close file
3. Revert to saved file
4. Delete file
5. Quit application
option?
```

Argumentet përmes komandave në reshta (Command Line Arguments)

Kur programi ekzekutohet në kuadër të një sistemi operativ (siç është DOS ose UNIX), ai mund të kalohet në zero ose më shumë argumente. Këto argumente shfaqen pas ekzekutimit të emrit të programit dhe janë të ndara me hapësira. Pasi që ato shfaqen në të njëjtin resht ku janë lëshuar komandat e sistemit operativ, ato thirren argumente të komandave në resht (command line arguments).

Si një shembull, konsideruar një program me emrin *sum* i cili printon[5] në dalje shumën e numrave futur në të si argumente të komandave në resht. Dialogu në vazhdim ilustron se si dy numra janë futur si argumente që të mblidhen ($ është komandë për UNIX).

[5] Printon – nxjerr në ekran pas ekzekutimit

```
$ sum 10.4 12.5
22.9
$
```

Argumentet të resht komandave janë bërë të lejueshme në C++ programin nga funksioni ***main***. Këtu janë dy mënyra në të cilat ***main*** mund të definohet.

```
int main (void);
int main (int argc, const char* argv[]);
```

E fundit përdoret kur programi është i mundur të pranojë argumente të resht komandave. Parametri i parë, *argc*, paraqet numrin e argumenteve të futura në program (duke përfshirë emrin e programit vetvetiu). Parametri i dytë, *argv*, është një vektor e stringjeve konstante që prezanton argumentet. Për shembull, në resht komandat në dialogun e mësimërm, kemi:

```
argc is 3
argv[0] is "sum"
argv[1] is "10.4"
argv[2] is "12.5"
```

Shembulli 8 ilustron një implementim të thjeshtë për *sum*. Stringjet janë konvertuar në numra real duke përdorur ***atof***, që është deklaruar në *stdlib.h*.

Shembull8:

```
#include <iostream.h>
#include <stdlib.h>

int main (int argc, const char *argv[])
{
        double sum = 0;
        for (int i = 1; i < argc; ++i)
                sum += atof(argv[i]);
        cout << sum << '\n';
        return 0;
}
```

Shembuj

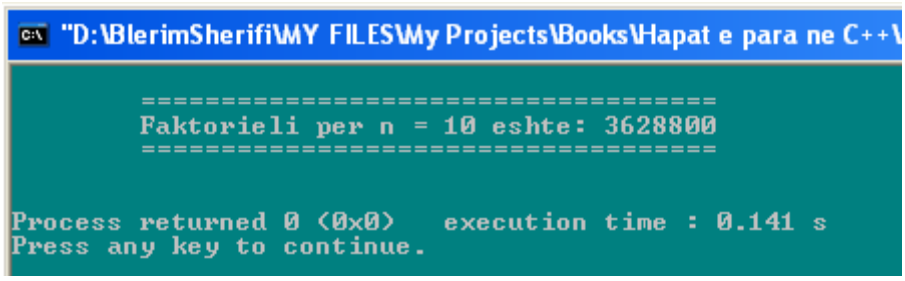
Në këtë kapitull nënkapitull do të ilustrojmë disa shembuj për funksionet që më së miri të kuptojmë përdorimin e tyre në program.

Shembull1: Të bëhet program në të cilin do të deklarohet një funksion me emrin faktorieli, i cili do të llogaris faktorielin në qoftë se *n = 10* dhe do të shfaq atë në ekran. Thirrja e këtij funksioni të bëhet nga programi kryesorë.

Zgjidhje:

```
// Eudhu bil-lahi minesh-shejtanir-raxhim
// Bismil-lahir-rahmanir-rahim
# include <iostream>
using namespace std;

void faktorieli ()
{
   int n=10;
   int faktorieli = 1;
   while (n>0)
   {
      faktorieli *= n;
      n--;
   }
   cout << faktorieli;
}
int main ()
{
   string vija = "\n\t====================================";
   cout << vija << "\n\tFaktorieli per n = 10 eshte: ";
   faktorieli();
   cout << vija << "\n\n";
   return 0;
}
```

Pas ekzekutimit të programit dalja në ekran do të duket si në vijim:



Shembull2: Të bëhet program për krahasimin e dy numrave dhjetorë:

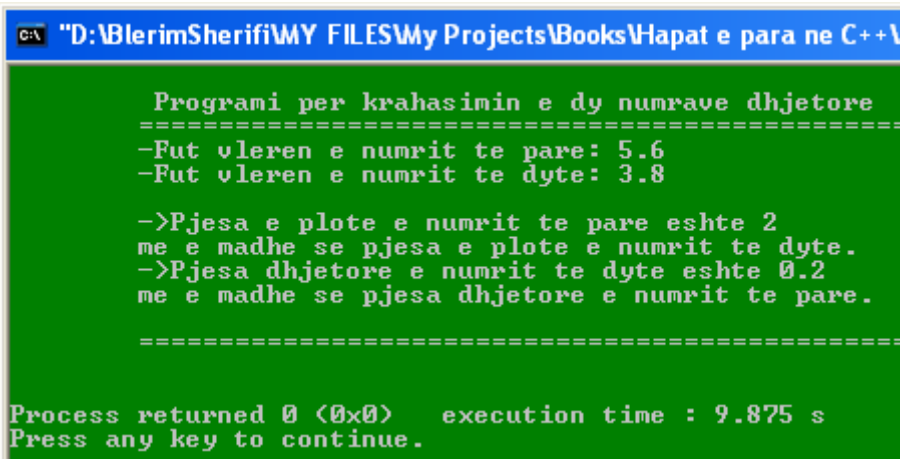
1. Në program të deklarohet një funksion me emrin krahasimi, që do të bën krahasimin e dy numrave dhjetorë.
2. Funksioni të përmbaj dy parametra për numrat që do të krahasohen.
3. Vlerat e dy numrave që do të krahasohen të jepen nga tastiera në funksionin kryesorë ku do të bëhet edhe thirrja e funksionit krahasimi.
4. Dalja në ekran të bëhet brenda funksionit krahasimi.

Zgjidhje:

```
// Eudhu bil-lahi minesh-shejtanir-raxhim
// Bismil-lahir-rahmanir-rahim
# include <iostream>
using namespace std;
void krahasimi (double ipari, double idyti);
int main ()
{
   string titulli = "\n\t Programi per krahasimin e dy numrave dhjetore";
   string vijat =
"\n\t=================================================";
   double ipari,idyti;
   cout << titulli << vijat;
   cout << "\n\t-Fut vleren e numrit te pare: ";
   cin >> ipari;
   cout << "\t-Fut vleren e numrit te dyte: ";
   cin >> idyti;
   krahasimi (ipari, idyti);          // Thirrja e funksionit krahasimi.
   cout << "\n"
        << vijat
        << "\n\n";
   return 0;
}
```

```
void krahasimi (double ipari, double idyti)
{
   int a = ipari;
   int b = idyti;
   int pjesa_e_plote;
   double pjesa_dhjetore;
   if (a > b)
   {
      pjesa_e_plote = a - b;
      cout << "\n\t->Pjesa e plote e numrit te pare eshte "
          << pjesa_e_plote
          << "\n\tme e madhe se pjesa e plote e numrit te dyte.";
   }
   else if (b > a)
   {
      pjesa_e_plote = b - a;
      cout << "\n\t->Pjesa e plote e numrit te dyte eshte "
          << pjesa_e_plote
          << "\n\tme e madhe se pjesa e plote e numrit te pare.";
   }
   else
      cout << "\n\t->Pjesa e plote e numrave eshte e barabarte.";
   if (ipari - a > idyti - b)
   {
      pjesa_dhjetore = (ipari - a) - (idyti - b);
      cout << "\n\t->Pjesa dhjetore e numrit te pare eshte "
          << pjesa_dhjetore
          << "\n\tme e madhe se pjesa dhjetore e numrit te dyte.";
   }
   else if (idyti - b > ipari - a)
   {
      pjesa_dhjetore = (idyti - b) - (ipari - a);
      cout << "\n\t->Pjesa dhjetore e numrit te dyte eshte "
          << pjesa_dhjetore
          << "\n\tme e madhe se pjesa dhjetore e numrit te pare.";
   }
   else
      cout << "\n\t->Pjesa dhjetore e numrave eshte e barabarte.";
}
```

Pas ekzekutimit të programit rezultati në ekran do të shfaqet si vijon:



Vërejtje: *Në këtë shembull kemi vetëm rastin kur pjesa e plotë e numrit të parë është më e madhe se pjesa e plotë e numrit të dytë, dhe pjesa dhjetore e numrit të dytë është më e madhe se pjesa dhjetore e numrit të parë. Për ta kuptuar më mirë ju jepni edhe raste tjera të hyrjeve p.sh., kur pjesa e plotë e numrit të parë është më e vogël se pjesa e plotë e numrit të dytë.*

Shembull3: Të bëhet program i cili do të llogarisë perimetrin e katërkëndëshit:

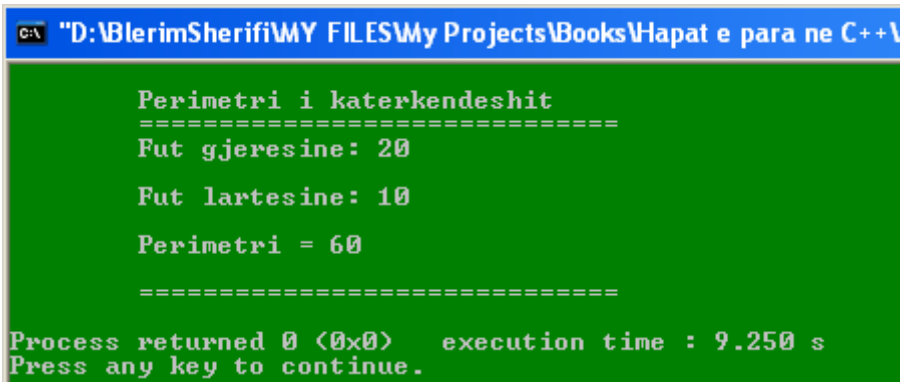
1. Të krijohet funksioni "katërkëndëshi" dhe në te të llogaritet perimetri.
2. Lartësia dhe gjerësia të futen përmes tastierës dhe ti jepen funksionit në fjalë, ndërsa ky i fundit të kthen vlerën e perimetrit.
3. Rezultati të nxirret në ekran.

Zgjidhje:

```
// ME EMRIN E ALL-LLAHUT MESHIRUESIT MESHIREBERESIT
# include <iostream>
using namespace std;
int katerkendeshi (int gjeresia, int lartesia);
int main ()
{
   string titulli = "\n\tPerimetri i katerkendeshit";
   string vijat = "\n\t==============================";
   int gj,l,P;
   cout << titulli << vijat;
   cout << "\n\tFut gjeresine: ";
   cin >> gj;
```

```
    cout << "\n\tFut lartesine: ";
    cin >> l;
    P = katerkendeshi(gj,l);
    cout << "\n\tPerimetri = "<<P<<endl<<vijat<<endl;
    return 0;
}
int katerkendeshi (int gjeresia, int lartesia)
{
    int Perimetri;
    Perimetri = 2 * gjeresia + 2 * lartesia;
    return Perimetri;
}
```

Pas ekzekutimit të programit, rezultati në ekran do të duket kështu:



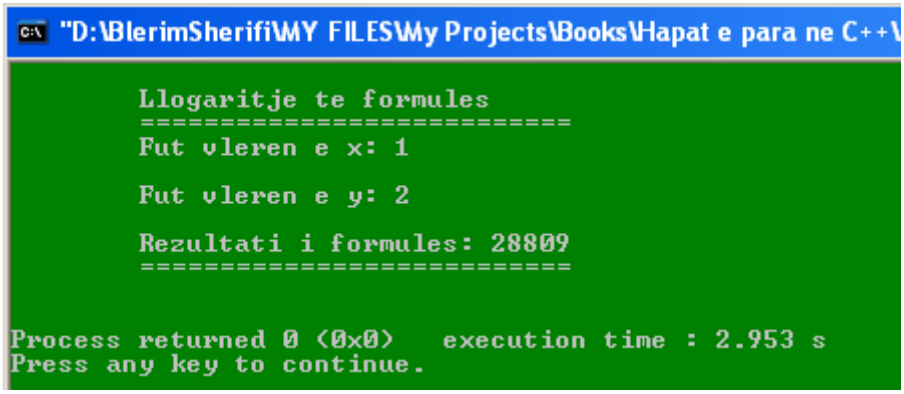
Shembull4: Të bëhet program i cili do të llogarisë këtë formulë :

1. Të krijohet funksion me emrin **"shuma"** që do të llogaris shumën, dhe
2. Funksioni me emrin **"prodhimi"** i cili do të llogaris prodhimin.
3. Vlerat për **x** dhe **y** të jepen nga tastiera, kurse vlera e **n** të jetë globale.
4. Rezultati të nxirret në ekran.

Zgjidhje:

```
// ME EMRIN E ALL-LLAHUT MESHIRUESIT MESHIREBERESIT
# include <iostream>
# include <math.h>
using namespace std;

int n = 10;         // ndryshore globale
double shuma (int x)
{
   int i;
   double shuma=0;
   for (i = 1; i <= n; i ++)
   {
      if (i != 7)
         shuma += pow(x,i);
   }
   return shuma;
}
int prodhimi (int y)
{
   int j, prodhimi=1;
   for (j = 1; j <= n; j++)
   {
      if (j!=4)
         prodhimi *= j/y+1;
   }
   return prodhimi;
}

int main ()
{
   string titulli = "\n\tLlogaritje te formules";
   string vijat = "\n\t============================";
   int x,y;
   double Shuma, Prodhimi;
   cout << titulli << vijat;
   cout << "\n\tFut vleren e x: ";
   cin >> x;
   cout << "\n\tFut vleren e y: ";
   cin >> y;
   Shuma = shuma(x);
   Prodhimi = prodhimi(y);
   cout << "\n\tRezultati i formules: "
        << Shuma + Prodhimi << vijat
        << "\n\n";
   return 0;
}
```

Pas ekzekutimit të programit rezultati në ekran do të jetë:



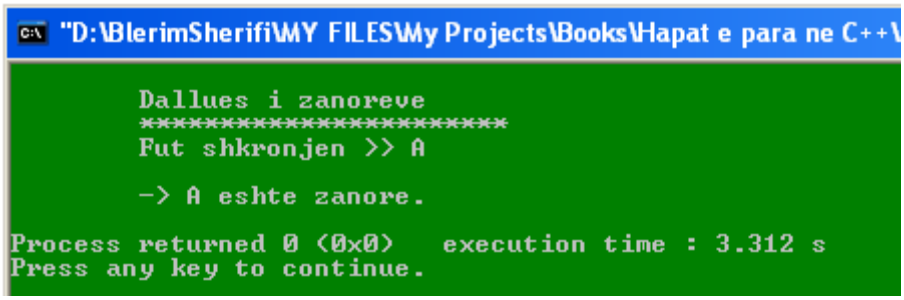
Shembull5: Të bëhet program i cili do të tregon se karakteri i futur nga tastiera a është zanore.

1. Karakteri të futet nga tastiera,
2. Krahasimi të bëhet në funksionin e quajtur "Zanore".
3. Rezultati të nxirret në ekran.

Zgjidhje:

```
// ME EMRIN E ALL-LLAHUT MESHIRUESIT MESHIREBERESIT
# include <iostream>
using namespace std;
int Zanore(char shkronja);
int main ()
{
   string titulli = "\n\tDallues i zanoreve";
   string vijat = "\n\t************************";
   char shkronja;
   cout << titulli << vijat;
   cout << "\n\tFut shkronjen >> ";
   cin >> shkronja;
   Zanore(shkronja);
   return 0;
}
int Zanore (char shkronja)
{
   switch (shkronja)
   {
      case 'a': case 'A':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      case 'e': case 'E':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      case 'i': case 'I':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      case 'u': case 'U':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      case 'ë': case 'Ë':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      case 'y': case 'Y':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      case 'o': case 'O':
      cout << "\n\t-> "<<shkronja << " eshte zanore.\n";
      break;
      default:
      cout << "\n\t-> "<<shkronja << " nuk eshte zanore.\n";
   }
}
```

Rezultati në ekran pas ekzekutimit të programit do të jetë:



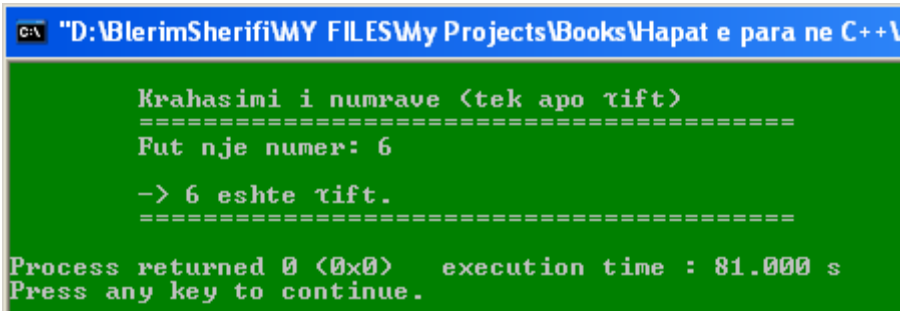
Shembull5: Të bëhet program i cili do të krahason numrin e futur nga tastiera është çift apo tek.

1. Të krijohet funksion me emrin krahasimi ku do të bën këtë krahasim.
2. Pastaj rezultati të nxirret në ekran.

Zgjidhje:

```
// ME EMRIN E ALL-LLAHUT MESHIRUESIT MESHIREBERESIT
# include <iostream>
using namespace std;
int krahaso (int numri);
int main ()
{
   string titulli = "\n\tKrahasimi i numrave (tek apo çift)";
   string vijat = "\n\t=======================================";
   int numri;
   cout << titulli << vijat;
   cout << "\n\tFut nje numer: ";
   cin >> numri;
   krahaso(numri);
   cout << vijat << endl;
   return 0;
}
int krahaso (int numri)
{
   if (numri%2==0)
      cout << "\n\t-> " << numri << " eshte çift.";
   else if (numri%2!=0)
      cout << "\n\t-> " << numri << " eshte tek.";
   else
      cout << "\n\tJu lutem futni ndonje numer.";
}
```

Pas ekzekutimit të programit dalja në ekran do të jetë:

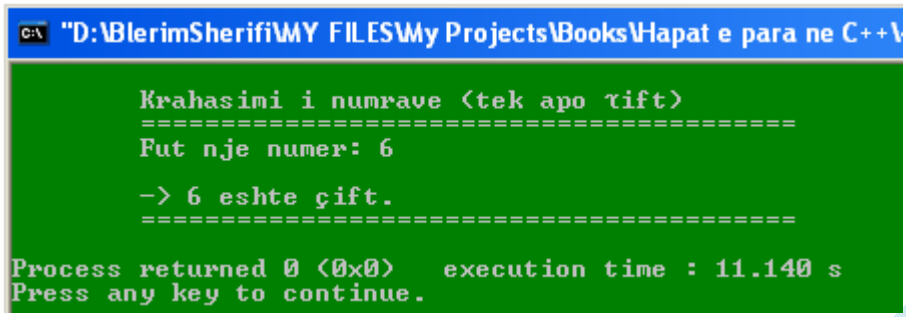


<u>Vërejtje:</u> Siç shihet në foton e më sipërme, shkronja ç nuk ka dalë por në vend të saj kemi një karakter tjetër. Kjo për arsye se kompjuteri nuk e njeh këtë shkronjë por duhet që në vend të kësaj shkronje të shënojmë kodin e saj 135 .

Më poshtë kemi një mënyrë për të shfaqur në ekran këtë shkronjë:

```
int krahaso (int numri)
{
   char a=135;       //deklarimi i një ndryshore karakter që mban kodin e shkronjës ç
   if (numri%2==0)
      cout << "\n\t-> " << numri << " eshte " <<a << "ift.";   // nxirret në ekran
                                                                 karakteri a
   else if (numri%2!=0)
      cout << "\n\t-> " << numri << " eshte tek.";
   else
      cout << "\n\tJu lutem futni ndonje numer.";
}
```

Dhe pasi të ekzekutojmë programin dalja do të duket kështu:



?Kontrollim Njohurish

Kapitulli VII

Fushat, Printerët dhe Referencat

Nënkapitujt:

- ✓ **Fushat (vektorët)**
- ✓ **Fushat shumëdimensionale (matricat)**
- ✓ **Pointerët (shënjuesit, treguesit)**
- ✓ **Memoria dinamike**
- ✓ **Pointerët aritmetikë**
- ✓ **Funksion pointerët**
- ✓ **Referancat**
- ✓ **Typedef komanda**
- ✓ **Shembuj**
- ✓ **Kontrollim njohurishë**

Fushat, Pointerët[6] dhe Referencat

Në këtë kapitull do të prezantojmë tre tipa të të dhënave: fusha, pointeri, referenca, dhe do të ilustrojmë përdorimin e tyre për përcaktimin e ndryshoreve.

Një fushë përbëhet nga një varg objektesh (të quajtura elementet e saja), ku të gjithë janë të tipit të njëjtë dhe janë të renditura në mënyrë të njëpasnjëshme në memorie. Në përgjithësi, vetëm fusha ka vetvetiu emër simbolik, jo elementet e saja. Çdo element është i përcaktuar me një indeks që paraqet pozicionin e elementit në atë fushë. Numri i elementeve në një fushë quhet *madhësia* e sajë. Madhësia e një fushe është fikse dhe e paracaktuar; ajo nuk mund të ndryshohet përderisa programi ekzekutohet.

Fushat janë të përshtatshme për reprezentimin e të dhënave të përbëra (përziera) që përmbajnë gjëra të ngjashme dhe individuale. Shembujt përfshijnë: një listë me emra, një tabelë me qytetet e botës dhe temperaturat e tyre, ose procesverbalet mujore për një llogari bankare.

Një *pointer* është thjeshtë adresa e një objekti në memorie. Përgjithësisht, objekteve mund t'iu qasemi në dy mënyra: drejtpërdrejt nga emri i tyre simbolik, ose indirekt nëpërmes pointerit. Veprimi për të arritur tek një objekt nëpërmjet pointerit, quhet ????????. Pointer ndryshoret definohen të çoj te objekti me tip specifik kështu kur pointeri të .....?., objekti tipik merret.

Pointerët janë të nevojshëm për krijim të objekteve dinamike përgjatë ekzekutimit të programit. Ndryshëm objektet normale (global dhe lokal) që janë caktuar të ruajtura në runtime stack, objekti dinamik është i caktuar në memorie nga një hapësirë memoruese e ndryshme e quajtur grumbull (ang. the heap[7]). Objektet dinamike nuk i zbatojnë rregullat normale të fushës (shtrirjes). Shtrirja e tyre kontrollohet hollësisht nga programuesi.

Referenca (adresimi) jep një emër alternativ simbolik (ang. alias[8]) për një objekt. Qasja e një objektit përmes referencës është saktësisht e njëjtë sikurse qasja përmes emrit të sajë origjinal. Referencat paraqesin fuqinë e shënjuesve (pointerëve) dhe lehtësi (komoditet) për qasje të drejtpërdrejtë tek objektet. Ata përdoren të ndihmojnë stilin e thirrjes-përmes-referencave të parametrave të funksioneve, veçanërisht kur objektet e mëdha janë të vendosura në funksion.

---

[6] Pointer – nga fjala anglese që në shqip do të thotë treguesit ose shënjuesit.
[7] Heap – grumbull, pirg.
[8] Alias – Pseudonim.

Fushat (vektorët)

Një varg ndryshores definohet duke specifikuar dimensionet e saja dhe tipin e elementeve të saja. Për shembull, një varg që prezanton 10 matje të gjatësisë (secila duke qenë një numër i plotë) mund të definohet si:

int heights[10];

Elementeve individuale të një vektori mund tu qasemi përmes indeksimit të rreshtave. Elementi i parë i vektorit gjithmonë e ka indeksin 0. Prandaj, **heights[0]** dhe **heights[9]** paraqesin elementin e parë dhe të fundit të vektorit *heights*. Secili nga elementet e vektorit *heights* mund të trajtohet si një ndryshore e plotë. Kështu, për shembull, për ta caktuar elementin e tretë 177, ne mund të shkruajm:

heights[2] = 177;

Përpjekja për të hyrë në një element që nuk ekziston në vektorë (p.sh., heights[-1] ose heights[10]) të con në një gabim të rëndë (i quajtur error: 'indeksi jashtë kufijve').

Përpunimi i një vektori zakonisht përfshin cikël që shkon nga elementi në element. Shembulli në vazhdim ilustron këtë duke përdorur një funksion i cili merr një vektor me numra të plotë dhe kthen mesataren e elementeve të tij.

Shembull:

```
const int size = 3;
double Average (int nums[size])
{
        double average = 0;
        for (register i = 0; i < size; ++i)
        average += nums[i];
        return average/size;
}
```

Ashtu siç ndryshoret tjera, edhe fushat mund të kenë inicializues. Kllapat gjarpëruese janë përdorur për ti specifikuar vlerat individuale të ndara me presje për elementet e fushës.

int nums[3] = {5, 10, 15};

kjo komandë inicializon tre elementet e fushës me emrin nums siw janë: 5, 10 dhe 15, përkatësisht. Kur numri i vlerave tek inicializuesi janë më pak se numri i elementeve, elementet e mbetura inicializohen me vlerë zero.

```
int nums[3] = {5, 10};          // nums[2] inicializohet në 0
```

Kur përdoret komplet inicializimi, madhësitë e fushës bëhen të tepërt, sepse numri i elementeve në inicializues është i padyshimtë. Definomi i parë i *nums* mund atypari të jetë ekuivalente e shkruar si:

```
int nums[] = {5, 10, 15};       // nuk nevojiten dimensione
```

Një tjetër situatë ku dimension mund të hiqet është për një parametër fushë të një funksioni. Për shembull *Average* funksioni i mësipërm mund të përmirësohet duke rishikuar atë ku dimensioni për *nums* nuk është më konstant, por e specifikuar nga një parametër shtesë. Shembulli në vazhdim ilustron këtë:

Shembull:

```
double Average (int nums[], int size)
{
        double average = 0;
        for (register i = 0; i < size; ++i)
        average += nums[i];
        return average/size;
}
```

C++ stringu është thjeshtë një fushë karakteresh. Për shembull,

```
char str[] = "HELLO";
```

definon *str* të jetë një fushë prej 6 karaktereve: gjashtë shkronja dhe një karakter null. Karakteri ndërprerës null është futur nga vet kompajleri. Nga ana tjetër,

```
char str[] = {'H', 'E', 'L', 'L', 'O'};
```

definon str të jetë një fushë me pesë karaktere.

Është e lehtë për të kalkuluar dimensionin e një fushe duke përdorur operatorin *sizeof*. Për shembull, duke pasur parasysh një fushë *ar* ku tipi i elementeve të sajë është *Type*, dimensioni i *ar* është:

sizeof(ar) / sizeof(Type)

Fushat shumëdimensionale (Matricat)

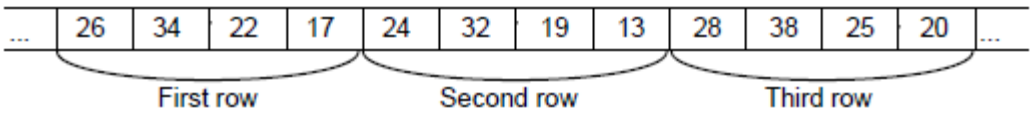
Një fushë mund të ketë më shumë se një madhësi (p.sh., dy, tre, ose më shumë). Organizimi i fushës në memorie ende është i njëjtë (sekuenca të një pas njëshme të elementeve), por të kuptuarit e programorëve për organizimin e elementeve është i ndryshëm. Për shembull, supozojmë se duam të prezantojmë temperaturën mesatare për tre qytetet më të mëdha të Australisë.

	Spring	Summer	Autumn	Winter
Sydney	26	34	22	17
Melbourne	24	32	19	13
Brisbarne	28	38	25	20

Kjo mund të prezantohet për mes fushës dy dimenzionale të numrave të plotë:

int seasonTemp[3][4];

Organizimi i kësaj fushe në memorie është si 12 numra të plotë të njëpasnjëshëm. Programori, prapëseprapë, mund të imagjinojë atë si tre reshta me nga katër numra të plotë për secilën. Figura ...



Si më parë, elementet janë të qasshme nga indeksimi i fushës. Indeks i veçantë nevojitet për çdo përmasë. Për shembull, temperatura mesatare e Sidneit (reshti i parë, kolona e dytë) jepet nga *seasonTemp[0][1]*.

Fusha mund të inicializohet nga inicializimi fole:

```
int seasonTemp[3][4] = {
                        {26, 34, 22, 17},
```

```
                                    {24, 32, 19, 13},
                                    {28, 38, 25, 20}
                                   };
```

Nga se kjo është hartuar në një fushë një dimenzionale prej 12 elementeve në memorie, kjo është ekuivalente me:

```
int seasonTemp[3][4] = {26, 34, 22, 17, 24, 32, 19, 13, 28, 38, 25, 20};
```

Inicializimi fole është i preferuar nga se më mirë bëhet më informative, kjo është më e gjithanshme. Për shembull, ajo bën të mundur inicializimin e vetëm elementit të parë të cdo reshti dhe ka mbështetje në vlerën fillestare zero (default to zero):

```
int seasonTemp[3][4] = {{26}, {24}, {28}};
```

Ne gjithashtu mund ta heqim madhësinë e parë (por jo madhësinë pasuese) dhe e lëmë atë të nxirret prej inicializimit:

```
int seasonTemp[][4] = {{26, 34, 22, 17},
                       {24, 32, 19, 13},
                       {28, 38, 25, 20}};
```

Procedimi i një fushe shumë dimenzionale është e ngjashme me një fushë një dimenzionale, por përdoren dy cikle në vend të një. Shembulli në vazhdim ilustron këtë duke shfaqur një funksion për gjetjen e temperaturës më të madhe në *seasonTemp*.

Shembull:

```
const int rows = 3;
const int columns = 4;
int seasonTemp[rows][columns] = {{26, 34, 22, 17},
                                 {24, 32, 19, 13},
                                 {28, 38, 25, 20}};

int HighestTemp (int temp[rows][columns])
{
        int highest = 0;
        for (register i = 0; i < rows; ++i)
                for (register j = 0; j < columns; ++j)
        if (temp[i][j] > highest)
        highest = temp[i][j];
        return highest;
}
```

Pointerët (treguesit)

Pointeri është thjeshtë adresa e lokacionit të memories dhe jep një rrugë indirekte për qasjen e të dhënave në memorie. Pointeri i ndryshores është definuar të "pikë në" (gisht në) të dhëna të tipave të veçantë. Për shembull:

int *ptr1; *// pointer në një numër të plotë (int)*
char *ptr2; *// pointer në një karakter (char)*

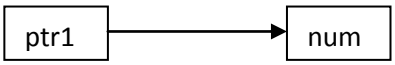
Vlera e ndryshores së pointuar është adresa në cilën ajo drejton. Për shembull, është dhënë definomi

int num;

ne mund të shënojmë:

ptr1 = #

Simboli **&** është operatori i adresës; ai merr ndryshoren si argument dhe kthen adresën e memories të asaj ndryshore. Efekti i caktimit të mësipërm është se adresa e *num* i është caktuar *ptr1*. Prandaj, ne thamë se *ptr1* pointon në *num*.
Shembulli në vazhdim ilustron këtë grafikisht



Një pointer i plotë i thjeshtë

Jep se *ptr1* shënon në *num*, shprehja

***ptr1**

....., dhe atypari është ekuivalente me *num*. Simboli ***** është operatori dereference; ai merr pointerin si argument dhe kthen përmbajtjen e lokacionit në të cilin ai është pointuar.

Në përgjithësi, tipi i pointerit patjetër të ngjasojë tipin e të dhënës në të cilën ai është i pointuar. Pointeri i tipit *void**, prapëseprapë, do të ngjasojë cilindo tip. Kjo është e dobishme për definimin e pointerëve që mund të pointojnë në të dhëna të tipave të ndryshme, ose tipa që kanë prejardhje të panjohur.

Pointeri mund të hidhet (tip i konvertuar) në një tip tjetër. Për shembull,

ptr2 = (char*) ptr1;

konverton *ptr1* në *char* pointer para se ti shoqërojë atë *ptr2*.

Pavëmendshëm në tipin e tij, pointeri mund të shoqërojë vlerën 0 (thirret null pointer). Pointeri **null** përdoret për inicializimin e pointerëve, dhe për shënimin e fundit të pointerit-bazë strukturave të të dhënave (p.sh., listat e linkuara).

Memoria dinamike

Në mbledhje të *raftit programorë*[9] (që përdoret për ruajtjen e ndryshoreve globale dhe stack frames për thirrjet e funksionit), tjetër hapësirë e memories, e quajtur *grumbull* (ang. heap), është e siguruar. Kjo hapësirë e memories përdoret për caktimin dinamik të blloqeve të memories përderisa programi ekzekutohet. Si rezultat, gjithashtu thirret *memorie dinamike*. Ngjashëm, edhe *rafti programor* gjithashtu thirret *memorie statike*.

Dy operatorë përdoren për ndarjen apo mos ndarjen e blloqeve të memories në hapësirën e memories të quajtur *grumbull*. Operatori *new*

[9] Rafti programorë – raft apo vend ku ruhet (grumbullohen) variablat globale të programit. Origjinalja në anglisht është *program stack*.

merr tipin si argument dhe i cakton blloqet e memories për një objekt të atij tipi. Kjo kthen pointer në blloqet e caktuara. Për shembull,

```
int *ptr = new int;
char *str = new char[10];
```

cakton, respektivisht, një bllok për ruajtjen e një numri të plotë dhe një bllok të gjatë mjaftueshëm për ruajtjen e një fushe me 10 karaktere.

Memorja e caktuar prej *grumbullit* nuk zbaton rregullat e njëjta të shtrirjes sikurse ndryshoret normale. Për shembull, në

```
void Foo (void)
{
        char *str = new char[10];
        //...
}
```

kur kthen Foo, ndryshorja lokale sur shkatërrohet, por blloku i memories i pointuar (shënjuar) nga str nuk është më. Ky i findit mbetet i caktuar derisa saktësisht të lirohet nga programori.

Operatori *delete* përdoret për lirimin e blloqeve memorie të caktuara nga *new*. Ky merr një pointer si argument dhe liron bllokun e memories në të cilin ai është i pointuar. Për shembull:

```
delete ptr; // delete an object
delete [] str; // delete an array of objects
```

Shënojmë se kur një bllok që fshihet është një fushë, si plotësim duhet të përfshihen kllapat [] për ta treguar këtë. Domethënia e kësaj vërehet kur në përdorim kemi klasat.

Delete do të aplikohet te pointeri që pointon në gjithçka por në mënyrë dinamike cakton objektet (p.sh., një ndryshore në raft), në këtë rast një gabim serioz mund të ndodhë gjatë kohës kur rrjedh programi (ang. runtime error). Është e padëmshme të përdoret *delete* në 0 pointer.

Objektet dinamike janë të dobishme për krijimin e të dhënave të cilat zgjasin përtej thirrjes së funksionit i cili krijon ato. Shembulli në vazhdim ilustron këtë duke përdorur një funksion që merr një parametër string dhe kthen kopjen e atij stringu.

```
#include <string.h>
char* CopyOf (const char *str)
{
        char *copy = new char[strlen(str) + 1];
```

```
        strcpy(copy, str);
        return copy;
}
```

Shënim

Reshti I: kjo është string kryefajlli standard që deklaron një shumëllojshmëri funksionesh për manipulimin me stringjet.

Reshti IV: funksioni *strlen* (deklaruar në *string.h*) numëron karakteret në atë string deri në karakterin e fundit null. Pasi që karakteri null nuk përfshihet në numërim, ne shtojmë 1 në total dhe caktojmë një fushë karakteresh të asaj madhësie.

Reshti V: funksioni *strcopy* (deklaruar në *string.h*) kopjon argumentin e tij të dytë për të parin, karakter për karakteri, edhe karakterin final null.

Për shkak të resurseve të kufizuara të kujtesës, gjithmonë është e mundur që memoria dinamike të jetë e rraskapitur përgjatë ekzekutimit të programit, sidomos kur shumë blloqe të mëdha janë të ndara dhe jo të liruara. *New* duhet të jetë i pa aftë të caktojë bllok të madhësisë së kërkuar, kjo do të kthen në vend 0. Është përgjegjësi e programuesit të merret me mundësi të tilla. Përjashtimi i mekanizmit të trajtimit të C++ jep metoda praktike të sjelljeve me probleme të tilla.

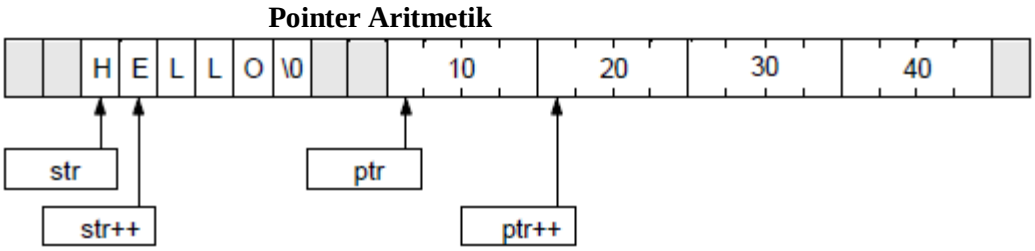
Pointer Aritmetikë

Në C++ mund të mbledhim një madhësi të plotë ose të zbresim një madhësi të plotë nga një pointer. Kjo është përdorur shpesh nga programuesit dhe quhet *pointer aritmetika.* Pointer aritmetika nuk është e njëjtë si aritmetika me numra të plotë, sepse rezultati varet nga madhësia e objektit pointuar në të. Për shembull, supozojmë se një *int* përfaqësohet nga 4 bajt. Tani, jepet

```
char *str = "HELLO";
int nums[] = {10, 20, 30, 40};
int *ptr = &nums[0];          // pointer në elementin e parë
```

str++ avancon *str* me një *char* (p.sh., një bajt) kështu që ajo pointon në karakterin e dytë të stringut “HELLO”, kurse *ptr++* avancon *ptr* me një

int (p.sh., katër bajt) kështu që ajo pointon në elementin e dytë të *nums*. Figura në vazhdim ilustron këtë skematikisht.



Ajo vjen, si pasojë, se elementet e "HELLO" mund të referohen si **str, *(str + 1), * (str + 2),* etc. Ngjashëm, elementet e *nums* mund të referohen si **ptr, *(ptr + 1), *(ptr + 2),* dhe ** (ptr + 3).*

Formë tjetër e pointimit aritmetik lejohet në C++ përfshirë zbritjen e dy pointerëve të tipit të njëjtë. Për shembull:

```
int *ptr1 = &nums[1];
int *ptr2 = &nums[3];
int n = ptr2 - ptr1;              // n bëhet 2
```

Pointer aritmetik është shumë praktik kur përpunohen elementet e një fushe. Shembulli në vazhdim shfaq shembull të një funksioni që kopjon string ngjashëm si *strcpy*.

```
void CopyString (char *dest, char *src)
{
        while (*dest++ = *src++)
        ;
}
```

Shënim

Reshti III: gjendja e këtij cikli cakton përmbajtjen e *src* në përmbajtjen e *dest* dhe pastaj rit bashkë të dy pointerët. Kjo gjendje bëhet 0 kur karakteri final null i *src* kopjohet në *dest.*

Rezulton se një ndryshore fushë (siç është nums) është vetvetiu adresa e elementit të parë e fushës që ajo prezanton. Kështu që elementet e *nums* mund gjithashtu të referohen të përdorin pointer aritmetikë në *nums*, ajo është, *nums[i]* është ekuivalente me **(nums + i)*. Ndryshimi në mes *num* dhe *ptr* është se *nums* është konstante, pra ajo nuk mund të

bëhet që të pointoj në diçka tjetër, kurse *ptr* është variabël dhe mund të bëhet që të pointon në ndonjë numër tjetër të plotë.

Shembulli i radhës shfaq si *HighestTemp* funksioni (i shfaqur më parë në shembullin ...) mund të përmirësohet duke përdorur pointerin aritmetik.

```
int HighestTemp (const int *temp, const int rows, const int columns)
{
    int highest = 0;
    for (register i = 0; i < rows; ++i)
    for (register j = 0; j < columns; ++j)
    if (*(temp + i * columns + j) > highest)
    highest = *(temp + i * columns + j);
    return highest;
}
```

Shënim

Reshti I: në vend të vendosjes së fushës në funksion, ne vendosim një *int* pointer dhe dy parametra plotësues që specifikojnë madhësitë të fushës. Në këtë mënyrë, funksioni nuk është i kufizuar në një madhësi të caktuar të fushës.

Rasti VI: shprehja **(temp + i * colums + j)* është ekuivalente me *temp[i][j]* në versionin paraprak të këtij funksioni.

HighestTemp mund të thjeshtësohet edhe më tej duke e trajtuar *temp* si një fushë një dimenzionale të *row * column* numra të plotë. Kjo është shfaqur në shembullin në vijim.

```
int HighestTemp (const int *temp, const int rows, const int
columns)
{
        int highest = 0;
        for (register i = 0; i < rows * columns; ++i)
        if (*(temp + i) > highest)
        highest = *(temp + i);
        return highest;
}
```

Funksion pointerët

Është e mundshme të marrim adresën e funksioni dhe të ruajmë atë në një funksion pointer. Pointeri pastaj mund të përdoret për të thirrur në mënyrë indirekte funksionin. Për shembull,

```
int (*Compare)(const char*, const char*);
```

definon një funksion pointer të emëruar *Compare* që mund të mbaj adresën e ndonjë funksioni që mer dy konstante pointer karakter si argumente dhe kthen një numër të plotë. Funksioni për krahasim të stringut në librari *strcmp*, për shembull, është i tillë.
Prandaj:

```
Compare = &strcmp;      //Compare pointon në funksionin strcmp
```

& operatori nuk është i domosdoshëm dhe mund të largohet:

```
Compare = strcmp;       //Compare pointon në funksionin strcmp
```

Në mënyrë alternative, pointeri mund të definohet dhe të inicializohet njëkohësisht:

```
int (*Compare)(const char*, const char*) = strcmp;
```

Kur një adresë funksioni caktohet në një funksion pointer, të dy tipat patjetër të përshtaten. Definomi i mësipërm është i vlefshëm sepse *strcmp* ka prototip të ngjashëm të funksionit:

```
int strcmp(const char*, const char*);
```

Duke pasur parasysh definimin e më sipërm të *Compare*, *strcmp* mund gjithashtu të thirret direkt, ose indirekt nëpërmjet *Compare*. Tre thirrjet pasuese janë ekuivalente:

```
strcmp("Tom", "Tim");          // thirrje direkte
(*Compare)("Tom", "Tim");      // thirrje indirekte
Compare("Tom", "Tim");         // thirrje indirekte (e shkurtuar)
```

Një përdorim i zakonshëm i funksion pointerit është të vendos atë si një argument tek një tjetër funksion; tipike, sepse kjo e fundit kërkon versione të ndryshme të më pashme në rrethana të ndryshme. Shembull i mirë është një funksion binar kërkimi për kërkim përmes një fushe të

renditura stringjesh. Ky funksion mund të përdor funksionin krahasues (siw është *strcmp*) për krahasimin e stringut kërkues përkrah stringjeve të fushës. Kjo ndoshta nuk mund të jetë e përshtatshme për të gjitha rastet. Për shembull, *strcmp* është *rast i ndjeshëm*. Nëse ne duam të bëjmë kërkim në mënyrë *jo rast të ndjeshëm* atëherë do të nevojitet funksion krahasues i ndryshëm.

E shfaqur në shembullin ..., duke bërë funksionin krahasues parametër të funksionit kërkues, ne mund të bëjmë këtë të fundit të pa varur nga ai i përparshmi.

Shembull:

```
int BinSearch (char *item, char *table[], int n,
        int (*Compare)(const char*, const char*))
{
        int bot = 0;
        int top = n - 1;
        int mid, cmp;
        while (bot <= top)
        {
                mid = (bot + top) / 2;
                if ((cmp = Compare(item,table[mid])) == 0)
                        return mid;       // kthen indeksin e item
                else if (cmp < 0)
                        top = mid - 1;    // kërkim  i kufizuar në pjesën e
                poshtme
                else
                        bot = mid + 1;    // kërkim i kufizuar në pjesën e sipërme
        }
return -1;                                // nuk u gjet
```

Shënim

Reshti I: kërkimi binar është një algoritëm i njohur për kërkimin nëpërmjet një liste artikuj të ruajtura. Lista e kërkimit është e paraqitur nga *tabela* që është një fushë stringjesh me n dimensione. Artikulli i kërkuar është i paraqitur nga *item.*

Reshti II: Compare është funksion pointer për tu përdorur për krahasimin e *item* përkrah elementeve të fushës.

Reshti VII: sa herë që rrotullohet cikli, hapësira e kërkimit përgjysmohet. Kjo përsëritet përderisa fundi i të dy hapësirave të kërkimit

(të paraqitura me *bot* dhe *top*) të ndeshet, ose përderisa ngjasimi të gjendet.

Reshti IX: artikulli krahasohet me pikën e mesme të fushës.

Reshti X: nëse *item* ngjason artikullin e mesëm, kthehet indeksi i këtij të fundit.

Reshti XI: nëse *item* është më i vogël se artikulli i mesëm, pastaj kërkimi kufizohet në gjysmën më të ulët të fushës.

Reshti XIV: nëse *item* është më i madh se artikulli i mesëm, atëherë kërkimi kufizohet në gjysmën e lartë të fushës.

Reshti XVI: kthen -1 të tregoj se këtu nuk ka artikull të ngjashëm.

Shembulli pasues shfaq si *BinSearch* mund të thirret me *strcmp* të vendosur si funksion krahasues:

```
char *cities[] = {"Boston", "London", "Sydney",
"Tokyo"};
cout << BinSearch("Sydney", cities, 4, strcmp) <<
'\n';
```

Kjo do të nxjerr 2 siç pritej.

Referencat

Një *referencë* prezanton një pseudonim për një objekt. Simboli për definimin e referencave është i ngjashëm me atë të pointerit, përveç se & është përdorur në vend të *. Për shembull,

```
double num1 = 3.14;
double &num2 = num1;      // num2 është
referencë në num1
```

definon *num2* si referencë për *num1*. Pas këtij definomi *num1* dhe *num2* të dyja i referohen objektit të njëjtë, sikurse ata të ishin ndryshore të njëjta. Duhet thekekuar se një referencë nuk krijon kopje të një objekti, por thjeshtë një pseudonim simbolik për të. Prandaj, pas

```
num1 = 0.16;
```

të dyja *num1* dhe *num2* do të tregojnë vlerën 0.16.

Referenca patjetër çdoherë të inicializohet kur ajo definohet: ajo duhet të jetë një pseudonim për diçka. Do të ishte e ndaluar të definohet një referencë dhe të inicializohet më vonë.

```
double &num3;          // e ndaluar: reference pa inicializues
num3 = num1;
```

Ju gjithashtu mund të inicializoni një reference në një konstante. Në këtë rast është bërë kopja e konstantes (pas ndonjë shndërrues tipi të nevojshëm) dhe referenca është vendosur të referoj në kopje.

```
int &n = 1;            // n referon në kopjen e 1
```

Arsyeja që *n* bëhet referencë në kopjen e *1* është se 1 vetvetiu është më i sigurt. Konsideroni se far mund të ndodhë nëse nuk do të ishte ky rast.

```
int &x = 1;
++x;
int y = x + 1;
```

Njëshi në reshtin e parë dhe njëshi në reshtin e tretë janë të mundshëm të bëhen objekti i njëjtë (shumë komajlerë bëjnë optimizmin e konstantes dhe vendosin të dy njëshat në të njëjtin lokacion të memories). Pra, edhe pse ne presim që *y* të bëhet 3, ajo mund të bëhet 3. Megjithatë, duke detyruar *x* të bëhet kopje e 1, kompajleri garanton se objekti i paraqitur nga *x* do të jetë i ndryshëm nga të dy njëshat.

Përdorimi më i zakonshëm i referencave është për parametrat e funksioneve. Parametrat referencë lehtëson stilin *pass-by-reference* së argumenteve, në krahasim me stilin *pass-by-value* që kemi përdorur deri më tani. Për të vëzhguar dallimet konsiderojmë tre funksione *swap* në shembullin në vijim.

Shembull:

```
void Swap1 (int x, int y)          // pass-by-value (objektet)
{
        int temp = x;
        x = y;
        y = temp;
}
void Swap2 (int *x, int *y)              // pass-by-value (pointeret)
{
        int temp = *x;
        *x = *y;
        *y = temp;
}
void Swap3 (int &x, int &y)              // pass-by-reference
{
        int temp = x;
        x = y;
        y = temp;
}
```

Shënim

Reshti I: edhe pse *Swap1* këmben x dhe y, kjo nuk ka efekt në argumentet e futura në funksion, sepse *Swap1* merr një kopje të argumenteve. Far ndodh në kopje nuk ndikon origjinali.

Reshti VII: Swap2 kapërcen problemin e *Swap1* duke përdorur parametrat pointer. Duke i respektuar printerët, *Swap2* kalon tek vlerat origjinale dhe shkëmben ato.

Reshti XIII: Swap3 kapërcen problemin e *Swap1* duke përdorur pointerët referencë. Parametrat bëhen pseudonime për argumentet të futura në funksion dhe i shkëmben ata si është menduar.

Swap3 ka një avantazh plus se sintaksa e thirrjes së tij është e njëjtë si *Swap1* dhe nuk përfshin adresim apo jo-referim. Funksioni *main* në vazhdim ilustron dallimet:

```
int main (void)
{
        int i = 10, j = 20;
        Swap1(i, j); cout << i << ", " << j << '\n';
         Swap2(&i, &j); cout << i << ", " << j << '\n';
        Swap3(i, j); cout << i << ", " << j << '\n';
}
```

Kur të ekzekutohet, do të prodhojë këtë rezultat:

10, 20
20, 10
10, 20

Typedef[10]

Typedef është një strukturë sintaktik për futjen e emrave simbolik për llojet e të dhënave. Ashtu si një referencë që përcakton një pseudonim për një objekt, një typedef përcakton një pseudonim për një tip. Përdorimi i sajë kryesorë është për të lehtësuar deklarimin e tipave të komplikuar si një ndihmë për të përmirësuar lejueshmërinë. Këtu janë disa shembuj:

```
typedef char *String;
Typedef char Name[12];
typedef unsigned int uint;
```

Efekti i këtyre definimeve është se *String* bëhet një pseudonim për *char**, *Name* bëhet një pseudonim për një fushë prej 12 karaktere, dhe *unit* bëhet një pseudonim për *unsigned int.* Prandaj:

```
String str;          // është e njëjtë si: char *str;
Name name;           // është e njëjtë si: char name[12];
uint n;              // është e njëjtë si: unsigned int n;
```

Deklarimi i komplikuar i *Compare* në shembullin 5.20 është kandidat i mirë për typedef:

```
typedef int (*Compare)(const char*, const char*);
int BinSearch (char *item, char *table[], int n, Compare
comp)
{
        //...
if ((cmp = comp(item, table[mid])) == 0)
        return mid;
        //...
}
```

[10] Typedef – sintaksë e gjuhës C++, që mundëson definim të tipit të të dhënave.

Typedef prezanton *Compare* si një emër tipi të ri për një funksion me prototip të dhënë. Kjo bën *BinSearch* bindshëm të thjeshtë.

Shembuj

Kontrollim Njohurish

Shembuj

Shembull1: Të bëhet një program për nxjerrjen në ekran të tekstit “Mirë se erdhe në C++.”.

Zgjidhja:

```
#include <iostream>
using namespace std;
int main ()
{
        cout >> “Mirë se erdhe në C++.”;   // dalja në ekran
        return 0;
}
```

Dalja në ekran do të jetë:

```
Mire se erdhe ne C++!

Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

Shembull2: Të bëhet një program për nxjerrjen në ekran të tekstit “Mirë se erdhe në C++.” Por fjalët e ndara të jenë në kryerresht.

Zgjidhja:

```
#include <iostream>
using namespace std;      // using namespace std; përdoret për të lexuar kompaileri
cout dhe cin

int main()                // programi kryesor
{                         // fillimi i trupit main
   cout << "Mire"
        << endl           // endl komandë për kalim në resht të ri
        << "se"
        << endl
        << "erdhe"
        << endl
        << "ne C++!"
        << endl;
   return 0;
}                         // mbarimi i trupit main
```

Komanda **endl** gjithashtu mund të zëvendësohet me komandën **\n**, e cila shënohet brenda apostrofave. Në vazhdim kemi shembullin2 me komandën **\n**:

```
#include <iostream>
using namespace std;

int main()
{
   cout << "Mire\nse\nerdhe\nne\nC++1\n";
   return 0;
}
```

Rezultati pas ekzekutimit tek të dy shembujt do të jetë i njëjtë:

```
Mire
se
erdhe
ne
C++1

Process returned 0 (0x0)   execution time : 0.234 s
Press any key to continue.
```

Shembull3: Të bëhet një program për futjen e një numri të plotë, dhe më pas leximin e atij numri.

Zgjidhja:

```
#include <isotream>
using namespace std;
int main ()
{
        int numri;                  // deklarimi i variablës numri
        cin >> numri; // futja e vlerës nga tastiera, që i shoqërohet variablës numri
        cout>> numri >> endl;  // dalja në ekran e numrit.
        return 0;
```

Dalja në ekran është:

```
5
5

Process returned 0 (0x0)   execution time : 8.080 s
Press any key to continue.
```

Në fillim fusim numrin 5 nga tastiera, pastaj i njëjti numër del në ekran.

Ky lloj programi mund të shkruhet ndryshe që të duket më qartë:

```
#include <iostream>
using namespace std;

int main()
{
   int numri;
   cout<<"Jep numrin: ";
   cin >>numri;

   cout<<"Numri i futur eshte: "
      <<numri
      <<endl;
   return 0;
}
```

Dalja në ekran do të duket si në foton në vijim:

```
Jep numrin: 5
Numri i futur eshte: 5

Process returned 0 (0x0)   execution time : 4.570 s
Press any key to continue.
```

Shembull4: Të bëhet program i cili do të mbledh dy numra të plotë, dhe në ekran të nxjerr shumën.

Zgjidhja:

```
#include <iostream>
using namespace std;

int mian()
{
   // deklarimi i numrit te pare
   int numri1;

   // deklarimi i numrit te dyte
   int numri2;

   // deklarimi i numrit te trete
   int shuma;
   // futja e numrit te pare
   cout << "Jep numrin e pare: ";
   cin >> numri1;

   // futja e numrit te dyte
   cout << "Jep numrin e dyte: ";
   cin >> numri2;

   // llogaritja e shumes
   shuma = numri1 + numri2;

   // nxjerja e shumes ne ekran
   cout << "Shuma= "
        << shuma
        << endl;
   return 0;
}
```

Deklarimi i variablave i të njëjtit tip siç është edhe në shembullin e lartshënuar mund të bëhet edhe në një resht, si më poshtë:

int numri1, numri2, shuma;

Shembull5: Të bëhet program që do të mbledh dy numra të futur nga tastiera dhe shumën do ta zbres me numrin e tretë të futur po ashtu nga tastiera.

Zgjedhja:

```
#include <iostream>
using namespace std;

int mian()
{
   // deklarimet
   int numri1, numri2, numri3, shuma, zbritja;

   // futja e numrit te pare
   cout << "Jep numrin e pare: ";
   cin >> numri1;
   // futja e numrit te dyte
   cout << "Jep numrin e dyte: ";
   cin >> numri2;


   // futja e numrit te trete
   cout << "Jep numrin e trete: ";
   cin >> numri3;

   // zbritja
   zbritja = shuma - numri3;

   // nxjerja e rezultatit ne ekran
   cout << "Rezultati= "
        << Zbritja
        << endl;

   return 0;
}
```

Rezultati në ekran do të duket kështu:

```
Jep numrin e pare: 3
Jep numrin e dyte: 4
Jep numrin e trete: 3
Rezultati= 4

Process returned 0 (0x0)   execution time : 16.209 s
Press any key to continue.
```

Shembull6: Të bëhet program për llogaritjen e kësaj formule 20/(5*2).

Zgjidhja:

```
#include <iostream>
using namespace std;

int main ()
{
   // llogaritja e formules 20/(5*2)

   //deklarimi i variables rezultati e tipit double
   double rezultati;

  //logaritja e formules
   rezultati = 20/(5*2);

   //nxerja e rezultatit ne ekran
   cout << "Rezultati = "
        << rezultati
        << endl;

   return 0;
}
```

Llogaritja bëhet në këtë mënyrë, pra njëjtë edhe si në matematikë, në fillim llogaritet shprehja brenda kllapave dhe më pas kryhet operacioni i pjesëtimit.
Dalja në ekran do të jetë:

```
Rezultati = 2

Process returned 0 (0x0)   execution time : 0.124 s
Press any key to continue.
```

Shembull7: Të shkruhet program për mbledhjen, shumëzimin dhe pjesëtimin e dy numrave të çfarëdoshëm të futur nga tastiera. Po ashtu rezultatet të nxirren në program.

Zgjidhja:

```
# include <iostream>
using namespace std;

int main()
{
   double numri1, numri2, shuma, ndryshimi, prodhimi,heresi;

   cout<<"Fut numrin 1: ";
   cin >> numri1;
   cout<<"Fut numrin 2: ";
   cin >> numri2;

   shuma = numri1 + numri2;
   ndryshimi = numri1 - numri2;
   prodhimi = numri1 * numri2;
   heresi = numri1 / numri2;

   cout << "Rezultatet:\n";
   cout << "\tShuma= " << shuma
        << "\n\tNdryshimi= " << ndryshimi
        << "\n\tProdhimi= " << prodhimi
        << "\n\tHeresi= " << heresi
        << endl;

   cout <<"\n\n";  // kalon kursorin dy herë në resht të ri
   return 0;
}
```

Rezultati pas ekzekutimit do të jetë:

```
Fut numrin 1: 3
Fut numrin 2: 5
Rezultatet:
        Shuma= 8
        Ndryshimi= -2
        Prodhimi= 15
        Heresi= 0.6


Process returned 0 (0x0)   execution time : 3.307 s
Press any key to continue.
```

Përmbajtja