Hyrje në C++

- **C++** është një gjuhë programimi me qëllime të përgjithshme.
- **C++** përdoret për krijimin e lojrave kompjuterike, aplikacione etj.
- **C++** është derivuar nga gjuha e programimit **C** dhe në pjesën më të madhe bazohet në të.

Mjedisi ekzekutimit

Kodi C++ mund të ekzekutohet, ruhet, dhe shpërndahet pa instaluar software shtesë.

Për të instaluar software shtësë në kompjuterin tuaj, duhen dy komponentët e mëposhtëm:

1. **Integrated Development Environment (IDE)**: siguron mjete p/r përshkrimin e kodit. Çdo tekst editor mund të përdoret si IDE.

2. **Kompilator**: Kompilon kodin burim në program të ekzekutueshëm. Janë disa kompilator C++ që mund të përdoren. Mund të përdorni mjete falas si **Code:Blocks**, i cili përfshin edhe IDE edhe kompilatorin i vlefshëm për sistemet operative Windows, Linux dhe MacOS.

Shënim: skedaret me kod burim C++ kanë si prapashtësë emërtimi .cpp, .cp ose .c .

Programi juaj i parë në C++:

Një program C++ është një grumbull komandash dhe instruksionesh.
Më poshtë paraqitet një kod i thjeshtë që afishon si output “Përshëndetje”:

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


int main()
{
cout << "Përshëndetje!";
return 0;
}
```

Le të shpjegojmë secilin prej instruksioneve të mësipërme:

#include <iostream>

C++ ofron koka(header) të ndryshme, të cilat nevojiten për mirëfunksionimin e programit. Programi i mësipërm kontrollon për header-in **<iostream>**. Simboli thurje **(#) në fillim të rreshtit** tregon kompiler-it pre-processor-et. Në këtë rast, **#include** tregon kokën pre-processorit kokën **<iostream>.** Koka **<iostream>** përcakton standartin të objekteve stream për të dhënat input dhe output.

Kompilatori C++ injoron rreshtat bosh, tab-et dhe hapsirat. Në përgjithësi, rreshtat bosh dhe hapsirat thjesht përmirësojnë lexueshmërinë dhe strukturën e kodit.

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

int main()
{
cout << "Përshëndetje!";
return 0;
}
```

using namespace std;

Në kodin e mësipërm rreshti **using namespace std;** i tregon kompilatorit të përdorë standartin **std** (standard) **namespace**, **std** namespace përfshin karakteristika të C++ Standard Library.

Funksioni Main

Programi fillon ekzekutimin me funksionin, **main()**.

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

int main()
{
```

```
cout << "Përshëndetje!";
return 0;
}
```

Kllapat tregojnë { } fillimin dhe mbarimin e funksionit, i cili quhet ndryshe edhe **trupi** i funksionit. Informacioni brenda kllapave, tregon çfarë realizon funksioni kur ekzekutohet.

Hyrja e çdo programi në C++ është funksioni **main()**.

Rreshti cout << "Përshëndetje!"; afishon 'Përshëndetje' në ekran.

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

int main()
{
cout << "Përshëndetje!";
return 0;
}
```

*Në C++, **stream-et** përdoren për veprime input dhe output.*

Cout

Në shumë mjedise programimi, standarti apriori për afishimin e rezulateve është ekrani. Në C++, **cout** është objekti stream që akseson ekranin.
cout përdoret në kombinim me operatorin e futjes së të dhënave (**<<**) për insertimin e të dhënave që vijnë pas tij në ekran. Operatori i insertimit mund të përdoret disa herë pas **cout**, për shembull:

```
cout << "Unë " << "po mësoj" << "C++!";
```

Në C++, pikëpresja **përdoret** për të treguar fundin e një instruksioni, ndaj çdo instruksion duhet të mbarojë me pikëpresje.

Instruksionet

Një bllok kodi është një bashkësi logjike instruksionesh, rreshuar nga kllapa hapëse dhe mbyllëse.

Për shembull:

```
{
cout << "Përshëndetje!";
return 0;
}
```

Ju mund të përdorni shumë instruksione në një rresht të vetëm, mjafton të jenë të ndara me pikëpresje, përndryshe do të marrim një error.

Kalimi në rresht të ri

Operatori **cout** nuk inserton një rresht të ri në fund të instruksionit.
Një mënyrë për printimin në dy rreshta është **endl,** i cili vendoset aty ku duhet thyerja e rreshtit:

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

int main()
{
cout << "Përshendetje!" << endl;
cout << "Unë po mësoj C++!";
return 0;
}
```

endl kalon tekstin e dytë në një rresht të ri.

Një mënyrë tjetër për kalimin në një rresht të ri është përdorimi i karaktereve **ën** si alternative në vend të **endl.**
Simboli backslash (ë) quhet karakteri arratisjes '**escape**', dhe tregon një karakter special.

Shembull:

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

int main()
{
cout << "Përshendetje! ën";
cout << "Unë po mësoj C++!";
```

```
return 0;
}
```

Përdorimi i dy “ën ën” do të krijonte një rresht të ri.

Afishimi në shumë rreshta

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

int main()
{
cout << "Përshendetje! ën Unë po mësoj ën C++!";
return 0;
}
```

Komentet

Komentet shpjegojnë arsyen e përdorimit të secilit prej kodeve.
Kompilatorët injorojnë çdo gjë që është brenda komenteve, prandaj komentet nuk shfaqen në rezultate.

Një koment që fillon me **(//) quhet ‘single-line’** koment, koment në një resht të vetëm. Simbolet // i tregojnë kompilatorit të injorojë çdo gjë që ndodhet pas deri në fund të rreshtit.

Për shembull:

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

int main()
{
// printon "Përshendetje!
cout << "Përshendetje!";
return 0;
}
```

Komentet që shkruhen në shumë rreshta fillojnë me simbolin **/*** dhe mbarojnë me simbolin ***/** duke u shkruar komente në një rresht të vetëm ose në shumë rreshta.

```
/* Ky është koment */

/*

C++ komentet mund të
shpërndahen në
 shumë rreshta

*/
```

Komentet mund të shkruhen kudo dhe mund të përsëriten shumë herë.
Brenda një komenti që ndodhet brenda simboleve **/*** dhe ***/,** // komentet nuk kanë asnjë kuptim, kjo lejon që të përdoren komentet brenda komenteve.

```
/*
cout << "Përshendetje!"; // printon "Përshendetje!

*/
```

Variablat

Krijimi i një variable, zë një një hapsirë të caktuar në memorje për të ruajtur vlerat.
Kompilatori, kërkon që të përcaktoni edhe tipin e të dhënave të çdo variable që deklaroni.

Për tipin e të dhënave numra të plotë përdoret fjala kyçe **int.**
Shembull, dhënia dhe printimi i vlerës s\ një variable:

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

int main()
{
int Variabla = 100;
cout << Variabla;
return 0;
}
// Output-i 100
```

Gjuha e programimit C++ është **case-sensitive**, që do të thotë **'variabla'** dhe '**Variabla'** janë dy identifikues të ndryeshëm.

Të gjitha variablat identifikohen me emrin dhe tipin e të dhënës para se të përdoren në program. Mund të përcaktohen disa variabla të të njëjtit tip njëherësh por të ndara me presje:

```
int a, b;  // përcaktimi i dy variablave të tipit int
```

Një variabël mund t'i jepet vlerë për t'u përdorur në veprime.

Për shembull, mund të krijojmë një variabël shtesë të quajtur shuma:

```
int a = 10;
int b = 15;
int shuma = a + b;
// Shuma = 25

int main()
{
int a = 10;
int b = 15;
int shuma = a + b;
cout << shuma;
return 0;
}
//Shuma
```

Shënim: *Variablat duhet të kenë gjithmonë* ***një tip dhe një vlerë*** *para se të përdoren në program.*

Deklarimi i variablave

Variablave mund t'u jepen vlerat në momentin e deklarimit ose më vonë.
Gjithashtu, mund të ndryshoni vlerën e një variable gjat\ programit.

Disa shembuj:

```
int a;
int b = 42;

a = 10;
b = 3;
```

Input-et e përdoruesit

Për të bërë të mundur një përdorues të shfaqë një vlerë input përdoret **cin** në kombinim me opratorin (>>).
Shembulli më poshtë tregon se si merren të dhënat nga përdoruesi dhe ruhen në variablën **num:**

```
int num;
cin >> num;
```

Programi më poshtë i kërkon përdoruesit të shkruajë një numër dhe e ruan atë në variablën **a**:

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

int main()
{
int a;
cout << "Ju lutem shkruani një numër /n";
cin >> a;

return 0;
}
```

Gjatë ekzekutimit të programit shfaqet mesazhi *"Ju lutem shkruani një numër"*, dhe pritet që përdoruesi të shkruajë një numër ose Enter, ose Return.
Vlera ruhet në variablën **a**.

Marrja e disa të dhënave gjatë programimit:

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

int main()
{
int a, b;
cout << "Shkruani një numër /n";
cin >> a;
cout << "Shkruani një numër /n";
cin >> b;

return 0;
}
```

Le të krijojmë një program që merr input dy numra dhe printon shumën e tyre.

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

int main()
{
int a, b;
int sum;
cout << "Shkruani një numër ën";
cin >> a;
cout << "Shkruani një numër ën";
cin >> b;
sum = a + b;
cout << "Shuma është: " << shuma << endl;

return 0;
}
```

Specifikimi i tipit të të dhënave kërkohet vetëm njëherë, në kohën e deklarimit të të dhënave. Pas kësaj, variabla mund të përdoret pa iu referuar tipit të të dhënës.

```
int a;
a = 10;
```

Specifikimi i të dhënës më shumë se njëherë **do të jap error**, ndërsa vlera e variablës mund të ndryshohet sa herë të jetë e nevojshme në një program.

Për shembull**:**

```
int a = 100;
a = 50;
cout << a;

// Output-i 50
```

Operatorët Aritmetikë

C++ suporton këto veprime aritmetike: **mbledhje, zbritje, shumëzim, pjestim dhe modul.**

Opertori i mbledhjes realizon mbledhjen e dy operandëve.

```
int x = 40 + 60;
cout << x;

// Output-i 100
```

Zbritja

Operatori i zbritjes realizon zbrintjen e dy operandëve.

```
int x = 100 - 60;
cout << x;

//Output-i 40
```

Shumëzimi

```
int x = 5 * 6;
cout << x;

//Output-i 30
```

Pjestimi

```
int x = 10 / 3;
cout << x;

// Output-i 3
```

Pjestimi me zero jep error.

Moduli

Operatori i modulit (%) kthen mbetjen e plotë pas pjestimit të dy numrave të plotë.

Për shembull:

```
int x = 25 % 7;
cout << x;

// Output-i 4
```

Operatorët e inkrentimit

Operatori i inkrementimit përdoret për rritjen e vlerës me një dhe përdoret nga operatori C++.

```
x++; //ekuivalent me x = x + 1
```

Për shembull:

```
int x = 11;
x++;
```

```
cout << x;
//Outputi 12
```

Operatori i inkrementimit ka dy versione: **prefix** dhe **postfix**.++x; //prefix
x++; //postfix
Prefix inkrementon vlerën dhe më pas kryen veprime me shprehjen.

Postfix kryenveprimet me shprehjen dhe më llogarit inkrementimin.

Shembull Prefix:

```
x = 5;
y = ++x;
// x është 6, y është 6
```

Postfix example:

```
x = 5;
y = x++;
```

Shembulli i **prefix** inkrementon vlerën e x, dhe më pas e shenjon atë në y.
Shembulli **postfix** shenjon vlerën e x tek y, dhe më pas e inkrementon atë.

Operatori Dekremetimit

Operaori i dekrementimit (--) punon në të njëjtën mënyrë si operatori i inkrementimit, por në vend të rritjes së vlerës, e zbret atë me një.

```
--x; // prefix
x--; // postfix
```

Krahasimet

Instruktori **if** përdoret për ekzekutimin e një kodi nëse një kusht është i vërtetë.

Sintaksa:

```
if (kushti){
//instruksionet
}
```

Kushti specifikon se cila shprehje do të vlerësohet. Nëse kushti është i vërtetë, do të ekzekutohen instruksionet brenda kllapave gjarpërueshe. Nëse kushti nuk është i vërtetë, instruksionet thjesht injorohen, dhe vazhdohet ekzekutimi i programimit që gjendet pas kushtit **if.**

Kushtet kontrollohen nga opratorët operacional.

Për shembull:

```
if (7 > 4) {
cout << "Po";
}

// Output-i "Po"
```

Kushti **if** kontrollon nëse (7>4), mqs është e vërtetë, ekzekuton instruktorin **cout.**
Kushti i përcaktuar brenda kllapave **if** nuk kërkon **pikëpresje.**

Operatorët Relacional

Operatori jo i barabartë ose i ndryshëm kontrollon nëse operandët janë të ndryshëm:

Për shembull:

```
if (10 != 10) {
cout << "Po";
}
```

Kushti i mësipërm nuk është i vërtetë prandaj blloku i kodit nuk ekzekutohet.

Shembull:

```
int a = 55;
int b = 33;
if (a > b) {
cout << "a është më e madhe se b";
}

// Output-i " a është më e madhe se b "
```

Instruksioni Else

Instruksioni **if** mund të ndiqet nga instruksioni **else**, i cili ekzekutohet kur **if** është **false**.
Sintaksa:

```
if (kushti) {
//instruksionet
}
else {
//instruksionet
}
```

Kompilatori do të testojë kushtet:
- Nëse kushti është i vërtetë, atëherë do të ekzekutohet trupi **if.**
- Nëse kushti nuk është i vërtetë, do të ekzekutohet trupi **else.**
Kur është vetëm një intruksion **if/else** mund të mos vendosen kllapat gjarpërueshe tek **else.**

Shembull:

```
int pike = 90;

if (pike < 50) {
cout << "Ju nuk kaloni." << endl;
}
else {
cout << "Ju kaluat." << endl;
}

// Output-i "Ju kaluat."
```

Mund të përdoren sa else të jetë e nevojshme.

Shembull:

```
int mark = 90;

if (pike < 50) {
cout << "Me vjen keq." << endl;
cout << "Ju nuk kaloni." << endl;
}
else {
cout << "Urime!" << endl;
```

```
cout << "Ju kaloni." << endl;
}

/* Output-i
Urime!"
Ju kaloni.
*/
```

Gjithashtu **if** mund të përdoret brenda **if-ve** të tjerë.

Për shembull:

```
int pike = 100;

if (pike >= 50) {
cout << "Ju kaluat." << endl;
if (pike == 100) {
cout <<"Perfekt" << endl;
}
}
else {
cout << "Ju deshtuat." << endl;
}

/*Output-i
Ju kaluat.
Perfekt!
*/
```

Në **C++** mund të përdoret një numër i pa limituar **if/else.**

Për shembull:

```
int mosha = 18;
if (mosha > 14) {
if(mosha >= 18) {
cout << "I rritur";
}
else {
cout << "Adoleshent";
}
```

```
}
else {
if (mosha > 0) {
cout << "Femije";
}
else {
cout << "Dicka eshte gabim";
}
}
```

Instruksioni if/else

Në instruksionet **if/else**, në një instruksion të vetëm mund të mos përdoren thonjëzat gjarpërueshe:

```
int a = 10;
if (a > 4)
cout << "Po";
else
cout << "Jo";
```

Gjithësesi përshirja e kllapave gjarpërueshe është metodë e mirë, sepse e bën kodin më të qartë dhe më të thjeshtë për t'u lexuar.

Ciklet e përsëritjes

Cikli i përsëritjes loop, ekzekuton një ose disa instruksione deri sa plotësohet një kusht i caktuar. Cikli i përsëritjes **while** ekzekuton në mënyrë të përsëritur një grup instruksionesh për sa kohë një kusht i caktuar është i vërtetë.

Sintaksa:

```
while (kushti) {
instruksioni(et);
}
```

Kur kushti nuk është më i vërtetë, programi ekzekton instruksionet që vijnë menjëherë pas instruksionit **while.**

Trupi i ciklit të përsëritjes **while**, është blloku i instruksioneve brenda kllapave gjarpërueshe.

Për shembull:

```
int num = 1;
while (num < 6) {
cout << "Numri: " << num << endl;
num = num + 1;
}

/* Output-i
Numri: 1
Numri: 2
Numri: 3
Numri: 4
Numri: 5
*/
```

Shembulli më sipër deklaron një variabël të barabartë me 1 (**int** num = 1). Cikli **while** kontrollon kushtin (num < 6), dhe ekzekuton instruksionet brenda trupit, duke rritur vlerën e variablës num me 1 sa herë intruksioni i përsëritjes ekzekutohet. Pas iteracionit të pestë, **num** bëhet 6, dhe kushti do të marrë vlerën false, duke ndaluar ekzekutimin.

Vlera e inkrementuar mund të ndryshohet, duke ndryshuar edhe numrin e herëve që ekzekutohet cikli:

```
int num = 1;
while (num < 6) {
cout << "Numri: " << num << endl;
num = num + 3;
}

/* Output-i
Numri: 1
Numri: 4
*/
```

Pa kushtin i cili pas disa iteracionesh mund të marrë vlerën **false**, cikli do të përsëritej në mënyrë të pafundme.

Përdorimi i inkrementimit dhe Dekremetimit

Operatorët si inkrementimi dhe dekrementimi mund të përdoren për të ndryshuar vlerat në cikël.

Për shembull:

```
int num = 1;
while (num < 6) {
cout << "Numri: " << num << endl;
num++;
}

/* Output-i
Numri: 1
Numri: 2
Numri: 3
Numri: 4
Numri: 5
*/

num++ është e njëvlershme me num = num + 1.
```

Përdorimi i ciklit while

Një cikël mund të përdoret për të marrë shumë input-e nga një përdorues.
Më poshtë paraqitet shembulli i një programi i cili merr 6 numra dhe e ruan atë në një variabël.

```
int num = 1;
int numri;
while (num <= 5) {
cin >> numri;
num++;
}
```

Kodi i mësipërm pyet përdoruesin 5 herë, dhe sa herë merret numri nga tastiera, ruhet në një variabël.

Më poshtë paraqitet llogaritja e shumës së numrave të marra nga përdoruesi.

```
int num = 1;
int numri;
int total = 0;

while (num <= 5) {
cin >> numri;
total += numri;
num++;
}
cout << total << endl;
```

Kodi i mësipërm shton numrin e dhënë nga përdoruesi në variablën total në çdo iteracion. Në momentin që ndalohet ekzekutimi i ciklit, printohet vlera e variablës **total** e cila është shuma e të gjithë variablave të insertuara nga përdoruesi.

Shënim: Variabla total ka një vlerë fillestare 0.

Cikli i përsëritjes For

Cikli **for** është një përsëritje e strukturës së kontrollit e cila ju lejon të shkruani një cikël i cili ekzekutohet prej një numër specifik herësh.

Sintaksa:

```
for ( inicializimi; kushti; inkrementimi) {
instruksioni(et);
}
```

Hapi i incializimit ekzekutohet i pari, dhe nuk përsëritet.
Më pas, kontrollohet kushti, nëse kushti është i vërtetë ekzekutohet trupi i instruksionit.

Në hapin tjetër, instruksioni i inkrementimit, rindryshon variablën e kontrollit të ciklit.
Më pas, trupi i ciklit përsërit veten, dhe ndalon vetëm kur kushti nuk është më i vërtetë.

Për shembull:

```
for (int x = 1; x < 10; x++) {
// disa instruksione
}
```

Instruksionet e inicializimit dhe inkrementimit mund të mos përdoren nëse nuk është e nevojshme, por pikëpresja është e domosdoshme.

Shembulli më poshtë përdor ciklin e përsëritjes **for** për printimin e numrave nga 0 deri në 9.

```
for (int a = 0; a < 10; a++) {
cout << a << endl;
}

/* Output-i
0
1
2
3
4
5
6
7
8
9
*/
```

Në hapin e inicializimit, ne deklarojmë një variabël a dhe e vendosim atë të barabartë me 0.
Ku kushti është a < 10.
Pas çdo iteracioni, ekzekutohet instruksioni a++ **.**
Kur inkrementohet a deri në 10, kushti kalon në false dhe cikli përsëritjes ndalon.

Instruksioni inkrementimit mund të ndryshohet.

```
for (int a = 0; a < 50; a+=10) {
cout << a << endl;
}
/* Output-i
0
10
20
30
40
*/
```

Gjithashtu edhe rasti i dekrementimit.

```
for (int a = 10; a >= 0; a -= 3) {
cout << a << endl;
}

/* Output-i
10
7
4
1
*/
```

Shënim: Kur përdorni ciklin **for**, mos harroni të përdorni pikëpresjet pas instruksioneve të inicializimit dhe kushtëzimit.

Cikli përsëritjes do...while

Ndryshe nga ciklet e tjera **for** dhe **while**, të cilat testojnë vërtetësinë e kushtit që në fillim cikli **do...while** kontrollon vërtetësinë e kushtit në fund të ciklit. Cikli **do...while** është i ngjashëm me ciklin **while**. Ndryshimi i vetëm është se cikli **do...while** ekzekuton ciklin të paktën njëherë.

Sintaksa:

```
do {
intruksioni(et);
} while (kushti);
```

Shembull:

```
int a = 1;
do {
cout << a << endl;
a++;
} while(a < 4);

/* Output-i:
1
2
```

```
3
*/
```

Shënim: Mos harroni pikëpresjen pas instruksionit **while**.

While vs. do...while

Nëse kushti vlerësohet në **false**, trupi i **do** do të ekzekutohet të paktën njëherë:

```
int a = 20;
do {
cout << a << endl;
a++;
} while(a < 3);

// Output-i 20
```

Funksionet

Një funksion është një grup instruksionesh për realizimin e një detyre të caktuar.

Ju mund të përcaktoni funksionet tuaja në C++.

Përdorimi i funksioneve ka disa avantazhe:
- Ju mund të ripërdorni kodin brenda funksionit.
- Nëse është e nevojshme ju mund të bëni ndryshime brenda një funksioni pa ndryshuar strukturën e të gjithë kodit.
- I njëjti funskion mund të përdoret për inpute të ndryshme.

Çdo program në C++ ka të paktën një funksion, funksionin **main()**.

Tipi kthimit të vlerave të funskionit

Funksioni **main** ka formën e përgjithshme si më poshtë:

```
int main()
{
// instruksionet
return 0;
}
```

Tipi kthimit të vlerave të funksionit përcaktohet para emrit të funksionit . Në shembullin e mësipërm, tipi i kthimit është **int**, pra funksioni kthen vlera into se numra të plotë. Zakonisht, një funksion kryen veprime duke mos kthyer asnjë vlerë. Funksione të tilla përcaktohen me fjalën **void**.

Përcaktimi i një funksioni

Sintaksa:

```
Tipi_i_kthimit emri_funksionit( lista parametrave )
{
trupi  i funksionit
}
```

tipi kthimit: Tipi i të dhënave që kthen funksioni.
emri funksionit: Emërtimi funksionit.
parametrat: Për kalimin e vlerave tek parametrat. Lista e parametrave i referohet tipit, radhës, dhe numrit të parametrave në funksion.
trupi i funksionit: Një ose disa instruksione që përcaktojnë se çfarë do të bëj funksioni.
Parametrat janë **opsional**; pra mund të përcaktoni funksione pa parametra.

Në shembullin më poshtë përcaktohet një funksion i cili nuk kthen asnjë vlerë, thjesht printon një rresht tekst në ekran.

```
void printo ()
{
```

```
cout << "Une po mesoj funksionet ne C++!";
}
```

Pra në funksionin e mësipërm nuk janë përcaktuar parametra dhe tipi i funksionit është void.

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


void printo() {
cout << "Une po mesoj funksionet ne C++!";
}

int main() {
printo ();

return 0;
}
```

Shënim: Thërritja e funksionit jashtë fusnkionit kryesor **main()** do të rezultojë error.

Deklarimi i një funksioni i tregon kompilatorit se si thërritet dhe se çfarë do të realizojë ky funksion.

Shembull:

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

//Deklarimi i funksionit
void printo();

int main() {
printo();

return 0;
```

```
}
```

Përcaktmi i funksionit:

```
void printo() {
cout << "Une po mesoj funksionet ne C++!";
}
```

Parametrat e funksioneve

Një funksion mund të ketë një ose disa parametra.

Shembull:

```
void printo(int x)
{
cout << x;
}
```

Përcaktimi i një funksioni i cili merr si parametër një numër të plotë dhe printon vlerën e tij.

Mjafton që parametrat e funksionit të përcaktohen njëherë dhe më pas mund t'i kalohen argumentat përkatës gjatë thërritjes së funksionit.

Shembull:

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


void printo(int x)
{
cout << x;
}

int main() {
printo(42);
```

```
}
// Output-i 42
```

Vlera 42 i kalohet funksionit si **argument**, dhe i kalohet **parametrit** të funksionit : **x**.
Të njëjtit funksion mund t'i kalohen argumenta të ndryshme.

Shembull:

```
int shumezimi_me_dy(int x) {
return x*2;
}
```

Funksioni i përcaktaur më lartë, merr si parametër një numër të plotë dhe kthen vlerën e tij të shumëzuar me dy.
Ky funksion mund të përdoret me argumenta të ndryshme.

```
int main() {
cout << shumezimi_me_dy (8);
// Output-i 16

cout << shumezimi_me_dy (5);
// Output-i 10

cout << shumezimi_me_dy (42);
// Output-i 84
}
```

Funksionet me shumë parametra

Për një funksion mund të përcaktohen disa parametra të ndara me presje.
Më poshtë paraqitet një shembull i thjeshtë i mbledhjes së dy parametrave.

```
int shuma(int x, int y) {
// trupi i funksionit

}
```

Për çdo parametër duhet të përcaktohet **tipi** dhe **emri i funksionit.**