Prof. Dr. Agni Dika
Material punues.
Nuk është për shumëzim!

02

Direktivat paraprocesorike

Njohuri të përgjithshme

Kompjuterët janë paisje elektronike digjitale të cilat i kuptojnë një numër të caktuar komandash, siç janë komandat për marrje të të dhënave përmes njësive hyrëse, ato për kryerje të operacioneve të ndryshme aritmetikore dhe logjike mbi të dhënat (përmes njësisë aritmetiko-logjike të procesorit), komandat për ruajtje të të dhënave në memorien periferike dhe për paraqitje të rezultateve në njësitë dalëse. Grumbulli i komandave të cilat i ipen kompjuterit me qëllim të zgjidhjes së një problemi duke u mbështetur në një algoritëm të caktuar quhet *program* (ang. program), kurse vetë procesi i shkruarjes së programit njihet si *programim* (ang. programming).

Puna e kompjuterit mbështetet në shfrytëzimin e *sinjaleve digjitale* (ang. digital signals), përmes të cilëve paraqiten dy *shifrat binare* (ang. binary digit) `0` dhe `1`, zakonisht si dy nivele të ndryshme tensionesh. Për këtë arsye, komandat të cilat i ipen kompjuterit, si dhe të dhënat që përpunohen brenda tij, paraqiten si *vargje të shifrave binare*. Kurse gjuha përmes së cilës komunikohet me kompjuterin njihet si *gjuhë e makinës* (ang. machine language). Shkruarja e programeve në gjuhën e makinës është një punë e rëndë, sepse duhet të mbahen në mend vargje të shifrave binare për komandat e veçanta si dhe të konvertohen të dhënat e zakonshme në një *kod binar* (ang. binary code).

Me qëllim të lehtësimit të programimit në gjuhën e makinës mund të shfrytëzohet *gjuha asembler* (ang. assembly language), te e cila komandat paraqesin shkurtesa ose *kode mnemonike* (ang. mnemonic code), kurse vlerat që shfrytëzohen gjatë llogaritjeve ipen direkt si numra heksadecimal, ose indirekt përmes emrave të variablave përkatëse. Kështu, p.sh. komandat për mbledhje, zbritje dhe shumëzim kompjuterit i jepen përmes shkurtesave `ADD`, `SUB` dhe `MULT`. Gjatë kësaj, këto komanda dhe vlerat numerike përcjellëse kompjuteri nuk i kupton direkt sepse nuk janë të shkruara në formë binare. Për këtë qëllim ato duhet të përkthehen në gjuhë të makinës duke e shfrytëzuar programin i cili quhet *Asembler* (ang. assembler).

Gjuha e makinës dhe gjuha asembler, meqë janë të afërta me logjikën e punës së kompjuterit, njihen si *gjuhë të ulta programuese* (ang. low-level languages). Por, me qëllim të lehtësimit të procesit të programimit, shfrytëzohen *gjuhë të larta programuese* (ang. high-level languages), të cilat janë të afërta me logjikën e punës së njeriut. Të tilla janë, p.sh. gjuhët `FORTRAN`, `BASIC`, `COBOL`, `C`, `PASCAL`, `C++`, `JAVA`, `C#` etj., ose versionet e ndryshme të tyre. Nënkuptohet se programet e shkruara në gjuhët e larta programuese nuk kuptohen direkt nga kompjuteri. Prandaj ato duhet të përkthehen në gjuhën e makinës duke e shfrytëzuar programin i cili njihet si *kompilues* (ang. compiler). Për programin e përkthyer në gjuhën e makinës thuhet se paraqet një *program objektiv* (ang. object program), ose një *kod objektiv* (ang. object code).

Programi i shkruar në gjuhën programuese C++ paraqet *program burimor* dhe si i tillë ruhet në një *fajll burimor*. Me procesin e kompilimit, kontrollohet saktësia e komandave të shkruara dhe, nëse nuk ka gabime, programi burimor përkthehet në *program objektiv*, gjë që shihet edhe në paraqitjen skematike e cila është dhënë në *Fig.1.1*.

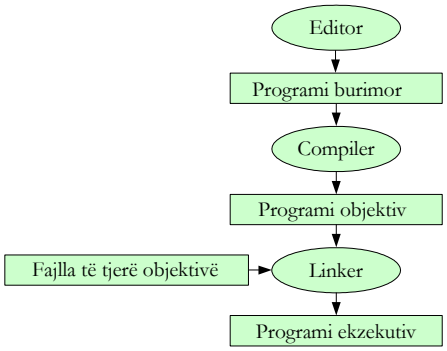

Fig.1.1
Procesi i krijimit të programit ekzekutiv nga programi burimor

Në fund, për ta fituar *programin ekzekutiv*, përmes *linkuesit* (ang. linker) fajllit objektiv i shtohen edhe fajlla të tjerë objektivë, të marra nga *biblioteka standarde* ose edhe prej bibliotekave tjera me fajlla. *Programi ekzekutiv* i cili fitohet si rezultat gjatë procesit në fjalë, ruhet në një *fajll ekzekutiv*.

Programet burimore në gjuhën C++ plotësohen edhe me funksione të domosdoshme që merren nga bibliotekat e shumta të kësaj gjuhe. Për këtë qëllim, në fillim të programeve vendosen të ashtuquajturit *fajlla të ballinës*, si *direktiva paraprocesorike* (ang. preprocessor directive), të cilat fillojnë me simbolin #. Inkorporimi i tyre në programet burimore, para kompilimit, bëhet përmes programit i cili quhet *paraprocesor* (ang. preprocessor). Paraqitja skematike e procesit të krijimit të programit ekzekutiv, në këtë rast duket ashtu siç është dhënë në *Fig.1.2*.

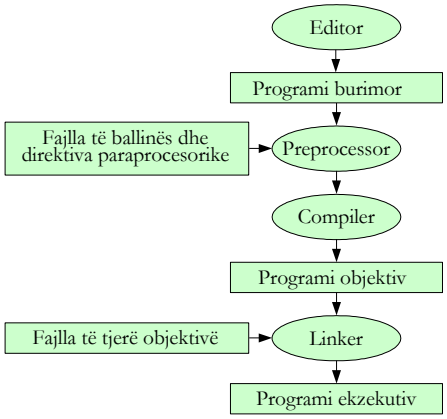

Fig.1.2
Versioni i plotësuar i procesit të krijimit të programit ekzekutiv

Direktivat paraprocesorike, për dallim nga komandat tjera që paraqiten në program fillojnë me simbolin #. Para kompilimit, përmes *paraprocesorit* këto direktiva insertohen brenda programit në pozitën përkatëse.

Në gjuhën programuese C++, fajllat burimorë zakonisht e kanë prapashtesën `.cpp`, fajllat e ballinës - prapashtesën `.h`, ose janë pa prapashtesë, fajllat objektiv - prapashtesën `.obj`, kurse fajllat ekzekutivë dallohen me prapashtesën `.exe`.

Pasi të përfundoj procesi i krijimit të programit ekzekutiv, përmes programit *Loader* (shih *Fig.1.3*) ai vendoset në memorien kryesore të kompjuterit. Në fund, si rezultat i punës së hardverit (ang. hardware) përkatës, programi ekzekutohet dhe i jep rezultatet e kërkuara.

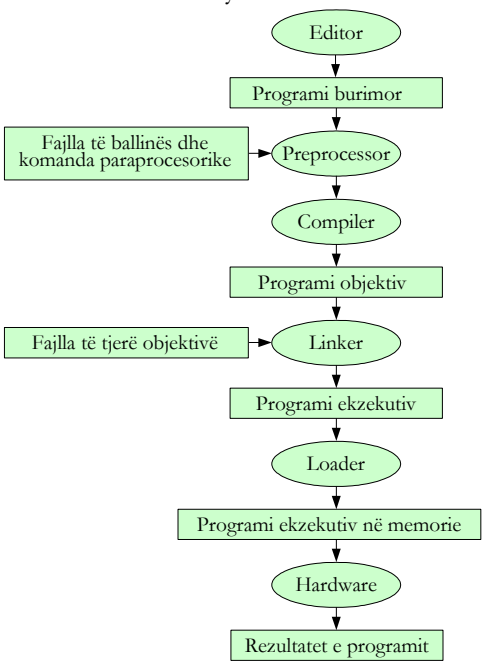

Fig.1.3
Versioni komplet prej shkruarjes deri te ekzekutimi i një programi në kompjuter

Paraprocesori

Rreshtat e programit të cilët fillojnë me simbolin **#** (ang. number sign, shqip thurje) përmbajnë *direktiva paraprocesorike* (ang. preprocessor directive), të cilat thjeshtë njihen edhe si *komanda paraprocesorike*. Këto direktiva përpunohen para kompilimit, përmes programit që quhet *paraprocesor* (ang. preprocessor), i cili është pjesë e kompilerit . Si rezultat, *programi burimor* modifikohet para se ai të kompilohet në *kod objektiv*. Këto direktiva nuk janë komanda të gjuhës programuese C++ dhe prandaj në fund të tyre nuk shënohet pikëpresja (`;`).

Paraprocesori është në gjendje t'i njohi direktivat praparocesorike të cilat janë përfshirë në tabelën e dhënë në *Fig.1.4*.

`#include`	`#define`	`#undef`	`#if`
`#ifdef`	`#ifndef`	`#elif`	`#else`
`#endif`	`#line`	`#pragma`	`#error`

Fig.1.4 Direktivat paraprocesorike

Në pjesën vijuese, përmes shembujve të programeve elementare do të tregohet shfrytëzimi i këtyre direktiva paraprocesorike.

Direktiva `#include`

Përmes direktivës paraprocesorike **`#include`** në programin burimor insertohet kopja e fajllit, i cili është shkruar paraprakisht dhe ruhet në disk. Zakonisht insertohen *fajlla standardë* të cilët gjenden në përbërje të pakos së gjuhës programuese C++, por insertohen edhe *fajlla jostandardë* të shkruar prej programerit.

Insertimi i fajllave standardë

Direktiva paraprocesorike e cila shfrytëzohet për insertimin e fajllave standradë shkruhet në formën:

```
#include <f>
```

Përmes kësaj komande në programin burimor insertohet kopja e fajllit me emër **`f`**, i cili është shënuar brenda *kllapave këndore* (ang. angle brackets), duke filluar prej rreshtit ku është vendosur kjo direktivë. E tillë p.sh. është direktiva:

```
#include <iostream>
```

e cila përdoret pothuajse në çdo program. Përmes saj në programin burimor insertohet fajlli `iostream` i klasës në të cilin mes tjerash përfshihen definicionet e komandave për lexim të vlerave hyrëse dhe shtypje të rezultateve.

Fajllat e tillë quhen edhe *fajlla të ballinës* (ang. header file), ose *fajlla insertues* (ang. include file). Forma e dhënë e kësaj direktive shfrytëzohet për fajllat që përfshihen në *bibliotekën standarde të fajllave të ballinës* (ang. standard library header files) dhe zakonisht vendosen në fillim të programit.

Insertimi i fajllave jostandardë

Në programin burimor mund të insertohen edhe *fajlla jostandardë*, të cilët shkruhen nga programeri. Për insertimin e fajllave të tillë shfrytëzohet direktiva paraprocesorike te e cila emri **`f`** i fajllit që insertohet shkruhet nën thonjëza:

```
#include "f"
```

Në këtë rast, emri i fajllit duhet të shoqërohet edhe me shtegun ku ai është vendosur në disk.

Fajlli që insertohet paraprakisht duhet të jetë shkruar përmes një tekstprocesori. Gjatë ruajtjes së tij në disk, përveç emrit fajlli mund të ketë edhe prapashtesë, p.sh. të formës **.txt**.

Shembull Programi përmes së cilit tregohet shfrytëzimi i direktivës paraprocesorike **#include**, për insertimin e fajllit **struct.txt**.

```
// Program include1
#include <iostream>
using namespace std;
#include "C:\Alfa\struct.txt"
int main()
{
    koha dita;
    dita.a=5;
    dita.b=dita.a+15;
    cout << "a="
         << dita.a
         << " b="
         << dita.b
         << endl;
 return 0;
}
```

Këtu, fajlli **struct.txt** është shkruar duke e shfrytëzuar programin *Microsoft Notepad* dhe ruhet në shtegun **C:\Alfa**. Përmbajtja e këtij fajlli është:

```
struct koha
{
    int a;
    double b;
};
```

Si rezultat i insertimit, para kompilimit pjesa e programit burimor duket kështu:

```
.....................
struct koha
{
    int a;
    double b;
};
int main()
{
```

```
    koha dita;
    dita.a=5;
    dita.b=dita.a+15;
    cout << dita.a
         << "   "
         << dita.b
         << endl;
return 0;
}
```

Për ta parë përmbajtjen e pjesës fillestare të programit të dhënë më sipër, pas insertimeve përmes direktivës paraprocesorike standarde **`#include <iostream>`**, duhet të shfrytëzohet doracaku i kompilerit përkatës.

Emri i fajllit mund të shënohet edhe nën kllapa këndore, kështu:

```
#include <C:\Test.txt>
```

Në këtë rast kompjuteri e kërkon fajllin në grupin e fajllave standard dhe pasi nuk e gjenë, vazhdon me kërkimin e tij në direktoriumin punues aktual, ose në direktoriumin që gjendet në shtegun e shkruar para fajllit (nëse ai është shënuar).

Shembull

Programi përmes së cilit në programin **`include2a`** insertohet fajlli **`alfa.txt`** i cili në veti e përmban definimin e funkcionit **`jeta`** për shtypje të një teksti.

```
// Programi include2a
#include <iostream>
using namespace std;
#include "C:\Alfa\dita.txt"
int main()
{
    jeta(1,10);
return 0;
}
```

Pjesa e programit që ruhet në fajllin **`dita.txt`** është:

```
void jeta(int m,int n)
{
    int i;
    for (i=m;i<=n;i++)
        cout << "Koha e bukur\n";
return;
}
```

Si rezultat, programi burimor pas insertimit të fajllit në fjalë duket kështu:

```
.....................
void jeta(int m,int n)
{
     int i;
     for (i=m;i<=n;i++)
         cout << "Koha e bukur\n";
return;
}
int main()
{
     jeta(1,10);
return 0;
}
```

Rez.

Në vijim është dhënë versioni i programit paraprak te i cili fajlli **dita.txt** insertohet në fund të tij.

```
// Programi include2b
#include <iostream>
using namespace std;
void jeta(int m,int n);
int main()
{
     jeta(1,10);
return 0;
}
#include "C:\Alfa\dita.txt"
```

Në këtë rast, programi burimor para kompilimit do të duket kështu:

```
.....................
void jeta(int m,int n);
int main()
{
     jeta(1,10);
return 0;
}
void jeta(int m,int n)
{
     int i;
     for (i=m;i<=n;i++)
         cout << "Koha e bukur\n";
return;
}
```

Direktivat paraprocesorike mund të vendosen kudo brenda programit. Gjithashtu, fajllat që insertohet nuk është e thënë që të përmbajnë tërësi funkcionale komplete të programeve.

Shembull

Versioni i programit paraprak te i cili direktiva paraprocesorike për insertimin e fajllit **delta.txt** është vendosur brenda programit.

```
// Programi include3
#include <iostream>
using namespace std;
int main()
{
    int i;
    #include "C:\Alfa\delta.txt"
return 0;
}
```

Nëse përmbajtja e fajllit **delta.txt** është:

```
for (i=1;i<=10;i++)
    cout << "Koha e bukur\n";
```

pas insertimit të tij, para kompilimit programi burimor duket kështu:

```
.................................
int main()
{
    int i;
    for (i=1;i<=10;i++)
        cout << "Koha e bukur\n";
return 0;
}
```

Rez. si edhe më parë.

Insertimi i një funkcioni për gjetje të vlerës maksimale të dy vlerave të dhëna.

Disa direktiva njëkohësisht

Në një program njëkohësisht mund të shfrytëzohen edhe disa direktiva paraprocesorike për insertim të fajllave jostandardë.

Shembull Programi në të cilin shfrytëzohen direktiva paraprocesorike për insertim të dy fajllave jostandardë **InpOut.txt** dhe **max.txt**.

```
// Program include2x
#include <iostream>
using namespace std;
int max(int,int);
int main()
{
     int x,y,z;
     #include "C:\Alfa\InpOut.txt"
     z=max(x,y);
     cout << "Vlera maksimale: "
          << z
          << endl;
return 0;
}
#include "C:\Alfa\max.txt"
```

Përmbajtjet e dy fajllave që insertohen janë:

- Fajlli **InpOut.txt**

```
cout << "Vlera x: ";
cin >> x;
cout << "Vlera y: ";
cin >> y;
```

- Fajlli **max.txt**

```
int max(int a,int b)
{
     int y;
     if (a>b)
          y=a;
     else
          y=b;
return y;
}
```

Këtu kemi të bëjmë me funkcionin **max** përmes së cilit definohet gjetja e vlerës maksimale mes dy numrave të dhënë **a** dhe **b**.

Programi, para kompilimit, pas insertimit të fajllave duket kështu:

```
.............................
int max(int,int);
int main()
{
     int x,y,z;
     cout << "Vlera x: ";
     cin >> x;
     cout << "Vlera y: ";
     cin >> y;
     z=max(7,4);
     cout << "Vlera maksimale: "
          << z
          << endl;
return 0;
}

int max(int a,int b)
{
    int y;
    if (a>b)
         y=a;
    else
         y=b;
return y;
}
```

Këtu duhet pasur kujdes gjatë insertimit të pjesëve të programit që variablat e pjesëve që insertohen **të mos vijnë në kundërshtim** me ato që egzistojnë në pogram (p.sh. inserimi i deklarimit të variablave, ose shfrytëzimi i variablave të cilat njëkohësisht paraqitne në pjesën që insertohen dhe në pjesën tjetër të programit).

Direktiva #define

Duke e shfrytëzuar direktivën **#define** mund të definohen *konstante simbolike*, ose *makro paraprocesorike* (ang. preprocessor macro). Përmes tyre programi bëhet më i lexueshëm, më i thjeshtë dhe mund të modifikohet më lehtë.

Në formë të përgjithshme kjo direktivë shkruhet kështu:

```
#define emri tekst
```

Gjatë përpunimit të kësaj direktive nga *paraprocesori*, programi burimor modifikohet para se ai të kompilohet në *kod objektiv* duke e zëvendësuar

emri-in me **tekst**-in, kudo që ai paraqitet në program. Këtu, **emri** përcaktohet si identifikator dhe e paraqet *emrin e konstantes simbolike* ose *emrin e makros*, kurse **tekst**-in paraprocesori *e sheh vetëm si tekst* dhe jo si diçka që është shkruar në gjuhën C++. Gjithashtu, identifikatorët e shfrytëzuar janë të veçant dhe paraqesin *simbole paraprocesorike* (ang. preprocessor symbol), sepse njihen vetëm nga paraprocesori.

Pavarsisht nga lloji, të gjitha njihen me një emër *makro*.

Konstantet simbolike

Në formën më të thjeshtë të saj, direktiva paraprocesorike **#define** shfrytëzohet për definimin e konstanteve simbolike.

Shembull

Programi për llogaritje të sipërfaqes dhe perimetrit të rrethit me rreze **r**, në të cilin përmes direktivës **#define** fillimisht definohet konstanta simbolike **pi** ku ruhet vlera e konstantes matematikore **3.1415926**.

```
// Programi define1
#include <iostream>
using namespace std;
#define pi 3.1415926
int main()
{
    double r,s,p;
    cout << "Rrezja e rrethit: ";
    cin >> r;
    s=pi*r*r;
    cout << "\nSipërfaqja e rrethit s="
         << s;
    p=2*pi*r;
    cout << "\nPerimetri i rrethit p="
         << p << endl;
return 0;
}
```

Paraprocesori në programin e dhënë e kërkon konstanten simbolike **pi** dhe sa herë që e gjenë atë e zëvendëson me konstanten numerike **3.1415926**. Si rezultat, kodi burimor para kompilimit duket kështu:

```
.....................................
int main()
{
    double r,s,p;
    cout << "Rrezja e rrethit: ";
```

```
    cin >> r;
    s=3.1415926*r*r;
    cout << "\nSipërfaqja e rrethit s="
         << s;
    p=2*3.1415926*r;
    cout << "\nPerimetri i rrethit p="
         << p << endl;
return 0;
}
```

Përmes direktivës **#define** mund të definohen edhe konstante simbolike të cilat përmbajnë tekste të çfardoshme.

Shembull

Programi për llogaritje të shumës së numrave natyror tek, ku përfshihet definimi i konstantes simbolike **alfa**, në të cilën ruhet teksti i shkruar nën thonjëza.

```
// Programi define
#include <iostream>
using namespace std;
#define alfa "Shuma e llogaritur: "
int main()
{
    double s=0;
    int i,n;
    cout << "Vlera hyrëse n: ";
    cin >> n;
    for (i=1;i<=n;i=i+2)
        s=s+i;
    cout << alfa
         << s
         << endl;
return 0;
}
```

Këtu, paraprocesori e zëvendëson konstanten simbolike **alfa** me tekstin përkatës dhe si rezultat para kompilimit, komanda e fundit për shtypje duket kështu:

```
    cout << "Shuma e llogaritur: "
         << s
         << endl;
```

Makrot paraprocesorike

Përveç për definimin e *konstanteve simbolike* të cilat u shpjeguan më sipër, direktiva **#define** mund të shfrytëzohen edhe për definimin e *makrove paraprocesorike*.

Makrot e zakonshëm

Nëse direktiva **#define** shfrytëzohet për definimin e komandave të gjuhës programuese C++, shprehjeve aritmetikore, pjesëve të ndryshme të programeve, ose edhe programeve komplete, mund të themi se kemi të bëjmë me *makro paraprocesorike të zakonshme*.

Definimi i komandave

Përmes direktivës në fjalë, në program si makro mund të definohen pjesë të komandave, ose edhe komanda komplete të gjuhës programuese C++.

Shembull Programi për llogaritje të shumës së numrave natyror tek, ku definohet makroja **shtyp** me komandën për shtypje **cout**.

```
// Programi define^
#include <iostream>
#define shtyp cout
using namespace std;
int main()
{
     double x,y,z;
     shtyp << "Vlera x=";
     cin >> x;
     shtyp << "\nVlera y=";
     cin >> y;
     z=-5;
     if (x<y)
          z=x+y;
     shtyp << "\nRezultati është z="
          << z
          << "\n";
return 0;
}
```

Këtu, paraprocesori e zëvendëson emrin simbolik **shtyp** me komandën **cout** kudo që ky emër paraqitet në program. Si rezultat, pas paraprocesimit, kodi burimor duket kështu:

```
.........................
```

```
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=-5;
     if (x<y)
          z=x+y;
     cout << "\nRezultati është z="
          << z
          << "\n";
return 0;
     }
```

Në makro mund të përfshihet edhe definimi i komandave me pjesët përcjellëse të tyre.

Shembull

Programi për llogaritje të *shumës së numrave natyror tek*, ku shfrytëzohet makroja **`lexo`** për definimin e tekstit **`cin >>`**.

```
// Programi define^
#include <iostream>
#define lexo cin >>
using namespace std;
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     lexo x;
     cout << "\nVlera y=";
     lexo y;
     z=-5;
     if (x<y)
          z=x+y;
     cout << "\nRezultati është z="
          << z
          << "\n";
return 0;
}
```

Në këtë rast, pas zëvendësimit të emrit simbolik **`lexo`** nga paraprocesori, komandat përkatëse në programin burimor duken kështu:

```
.........................
int main()
```

```
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
.........................
```

Shembull

```
// Shfrytezimi i #define
# include <iostream>
using namespace std;
enum viti
{
     pranvera, vera, vjeshta, dimri
};
#define tekst "\nStina e zgjedhur %sht% "
#define shtyp cout <<
int main()
{
     viti stina;
     stina=vera;
     shtyp tekst;
     switch (stina)
     {
     case pranvera:
          shtyp "pranvera";
          break;
     case vera:
          shtyp "vera";
          break;
     case vjeshta:
          shtyp "vjeshta";
          break;
     case dimri:
          shtyp "dimri";
     }
     cout << "\n\n";
return 0;
}
```

Definimi i shprehjeve metematikore

Makrot mund të shfrytëzohen edhe për definimin e shprehjeve të ndryshme matematikore.

Shembull Programi në të cilin shfrytëzohet makroja **delta** për definimin e shprehjes matematikore **z=2*x+y**, si dhe makrot **shtyp** e **lexo** në format e dhëna më shembujt paraprak.

```
// Programi
#include <iostream>
using namespace std;
#define delta z=2*x+y
#define shtyp cout <<
#define lexo cin >>
int main()
{
     double x,y,z;
     shtyp "Vlera x=";
     lexo x;
     shtyp "\nVlera y=";
     lexo y;
     delta;
     shtyp "\nRezultati është z=";
     shtyp z
          << "\n";
return 0;
}
```

Pas zëvendësimit të emrave simbolik **delta**, **shtyp** dhe **lexo** nga paraprocesori, programi burimor duken kështu:

```
.....................................
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=2*x+y;
     cout << "\nRezultati është z=";
     cout << z
          << "\n";
return 0;
}
```

Këtu, edhe pikpresja e shënuar në fund të shprehjes matematikore mund të përfshihet në makro, kështu:

```
#define delta z=2*x+y;
```

Pas kësaj, gjatë shfrytëzimit të makros në program, pikpresja nuk shënohet pas emrit **delta**.

Definimi i makros në disa rreshta

Nëse në makro definimi kërkon më shum rreshta, për vazhdimin në rreshtin vijues shfrytëzohet vija e pjerrët (\), pas së cilës kalohet në rresht të ri duke e shtypur pullën *Enter*, pa lënë asnjë zbrazësirë.

Shembull Programi në të cilin shfrytëzohet makroja **llogaritja** për definimin e një vargu komandash që shkruhen në disa rreshta.

```
// Programi
#include <iostream>
using namespace std;
#define llogaritja          \
             if (x<y)       \
                 z=x+y;     \
             else           \
                 z=2*y;
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     llogaritja
     cout << "\nRezultati është z=";
     cout << z
          << "\n";
return 0;
}
```

Pasi paraprocesori ta zëvendësoj emrin simbolik **llogaritja** me shprehjen e përfshirë në makron përkatëse, programi burimor para kompilimit duken kështu:

```
.....................................
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
```

```
     if (x<y)
         z=x+y;
     else
         z=2*y;
     cout << "\nRezultati është z=";
     cout << z
          << "\n";
return 0;
}
```

Makrot në brendi të programit

Definimi i makrove nuk është e thënë të bëhet në fillim të programit. Ato mund të definohen kudo brenda tij, por *patjetër para se të shfrytëzohen*.

Shembull

Programi në të cilin makroja **`llogaritja`** nga detyra paraprake është vendosur brenda programit.

```
// Programi
#include <iostream>
using namespace std;
int main()
{
     double x,y,z;
     #define llogaritja   \
         if (x<y)         \
             z=x+y;       \
         else             \
             z=2*y;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     llogaritja
     cout << "\nRezultati është z=";
     cout << z
          << "\n";
return 0;
}
```

Edhe në këtë rast efekti i veprimit të paraprocesorit është i njëjtë si edhe në detyrën paraprake dhe pamja e programit burimor para kompilimit është e njëjtë.

Definimi i komplet programit

Përmes një makroje mund të definohet edhe komplet programi.

Shembull

Shfrytëzimi i makros **`programi`** përmes së cilit definohet komplet programi për gjetjen e vlerës më të madhe të dy numrave që ruhen te variablat **`x`** dhe **`y`**.

```
// Programi defineXXXX
#include <iostream>
using namespace std;
#define programi                          \
int main()                                \
{                                         \
     double x,y,z;                        \
     cout << "Vlera x=";                  \
     cin >> x;                            \
     cout << "\nVlera y=";                \
     cin >> y;                            \
    z=(x>y) ? x : y;                      \
    cout << "\nVlear maksimale: "         \
         << z                             \
         << "\n";                         \
return 0;                                 \
}
programi
```

Në këtë rast, kodi burimor para kompilimit duket:

```
.................................
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=(x>y) ? x : y;
     cout << "\nVlear maksimale: "
          << z
          << "\n";
return 0;
}
```

Makro të paremtrizuara

Përmes direktivës **#define** mund të definohen edhe makro me argumente, ose siç quhen ndryshe *makro të parametrizuar*. Meqë makrot e tilla u përngjajnë funkcioneve, ato njihen edhe si *makro funkcione* (ang. function macros).

Shembull

Programi në të cilin definohet makroja **max** përmes së cilit gjendet vlera më e madhe mes dy parametrave **a** dhe **b** të cilët janë shënuar brenda kllapave.

```
// Programi
#include <iostream>
using namespace std;
#define max(a,b) ((a)>(b)) ? (a) : (b)
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=max(x,y);
     cout << "Vlera maksimale: "
          << z
          << endl;
return 0;
}
```

Këtu, paraprocesori në kodin burimor e kërkon sekuencën e simboleve **max(x,y)** dhe kur e gjenë, atë e zëvendëson me shprehjen:

```
(x>y) ? (x) : (y)
```

Si rezultat, pas paraprocesimit kodi burimor duket kështu:

```
.............................
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=(x>y) ? (x) : (y);
```

```
     cout << "Vlera maksimale: "
          << z
          << endl;
return 0;
}
```

Kjo është e ngjashme me funkcionin e tipit **inline**, i cili në program mund të duket ashtu siç është dhënë në vijim.

```
// Programi define1
#include <iostream>
using namespace std;
inline int max(int a,int b)
{
    return ((a)>(b)) ? (a) : (b);
}
int main()
{
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=max(x,y);
     cout << "Vlera maksimale: "
          << z
         << endl;
return 0;
}
```

Te makrot duhet pasur kujdes që variablat që përfshihen në shprehjen e saj të shkruhen brenda kllapave, sepse rez. mund të jetë i pasaktë.

Shembull

```
// Programi
#include <iostream>
using namespace std;
#define alfa(a) a/2
int main()
{
     int x,y;
     cout << "Vlera x=";
     cin >> x;
     y=alfa(x+4);
     cout << "Vlera e llogaritur: "
```

Këtu, nëse pas ekzekutimit të programit si vlerë hyrëse për variablën x e japim numrin 2, kompjuteri si rez. e jep numrin e gabuar 4 sepse llogaritë me shprehjen:

```
y=x+4/2=2+2=4
```

Nëse makroja shkruhet sakt (me kllapa):

```
#define alfa(a) (a)/2
```

gjatë llogaritjes, për vlerën hyrëse x=2 në këtë rast e kemi vlerën e saktë:

```
y=(x+4)/2=(2+4)/2=3
```

Një makro të parametrizuar brenda programit mund të shfrytëzohet edhe disa herë.

Shembull

```
// Programi
#include <iostream>
using namespace std;
#define prod(a,b) ((a)*(b))
int main()
{
     int x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     z=3*prod(x,x+2*y)+2*prod(x+1,3*x);
     cout << "Vlera e llogaritur: "
          << z
          << endl;
return 0;
}
```

Pas paraprocesimit, shprehja për **z** është:

```
z=3*(x)*(x+2*y)+2*(x+1)*(3*x);
```

Si parametër i makros mund të merret edhe rezultati i llogaritjes së definuar.

Shembull

```
// Programi define88
#include <iostream>
#define alfa(a,b,g)         \
             if (a<b)       \
                 g=a+b;     \
             else           \
                 g=2*a+b;
using namespace std;
int main()
{
    double x,y,z;
     cout << "Vlera x=";
     cin >> x;
     cout << "\nVlera y=";
     cin >> y;
     alfa(x,y,z)
     cout << "\nRezultati z="
          << z
          << endl;
return 0;
}
```

Makro të ndërthurura

Gjatë definimit të makrove mund të shfrytëzohen variabla të makrove që janë definuar paraprakisht, për të fituar *makro të ndërthurura*.

Shembull

```
// Programi define=====
#include <iostream>
using namespace std;
#define k x>=y
#define baraz =
int main()
{
#define alfa                            \
             if (k)                      \
                 z baraz 2 * x + y;     \
             else                        \
                 z baraz x + 3 * y;
     double x,y,z;
     cout << "Vlera x=";
     cin >> x;
```

```
     cout << "\nVlera y=";
     cin >> y;
     alfa
     cout << "\nRezultati z="
          << z
          << endl;
return 0;
}
```

Këtu duhet pasur kujdes që gjatë shfrytëzimit të identifikatorit **baraz**, para dhe pas tij të lihet së paku një zbrazësirë.

Shembull Makroja **programi** që është dhënë më parë gjatë definimit të së cilit thirren makrot **shtyp** dhe **lexo**.

```
// Programi defineyyy
#include <iostream>
#define shtyp cout <<
#define lexo cin >>
using namespace std;
#define programi                          \
int main()                                \
{                                         \
     double x,y,z;                        \
     shtyp "Vlera x=";                    \
     lexo x;                              \
     shtyp "\nVlera y=";                  \
     lexo y;                              \
     z=(x>y) ? x : y;                     \
     shtyp "\nVlear maksimale: "          \
          << z                            \
          << "\n";                       \
return 0;                                 \
}
programi
```

Njëlloj mund të ndërthuren makrot e zakonshme edhe me makro parametrike.

Shembull Makroja **programi** që është dhënë më parë gjatë definimit të së cilit thirren makrot **shtyp** dhe **lexo**.

```
// Programi define=====
```

```
#include <iostream>
using namespace std;
#define pi 3.1415926
#define siper(r) pi*r*r
int main()
{
    double r,s,p;
    cout << "Rrezja e rrethit: ";
    cin >> r;
    s=siper(r);
    cout << "\nSipërfaqja e rrethit s="
         << s;
    p=2*pi*r;
    cout << "\nPerimetri i rrethit p="
         << p << endl;
return 0;
}
```

Direktiva #undef

Duke e shfrytëzuar direktivën `#undef` mund të eliminohet efekti i definimit me direktivën **`#define`**. Një makro vlenë derisa nuk eliminohet përmes direktivës `#undef`.

```
// Programi undef
#include <iostream>
using namespace std;
#define n 3
int main()
{
    int i,j;
    cout << "\ni=";
    for (i=1;i<=n;i++)
        cout << i
             << "  ";
#undef n
#define n 5
    cout << "\nj=";
    for (j=1;j<=n;j++)
        cout << j
             << "  ";
    cout << endl;
return 0;
}
```

Nëse menjëherë pas direktivës:

```
#undef n
```

e shkruajmë komandën:

```
cout << n;
```

gjatë kompilimit do të paraqitet mesazhi *undeclaret identifier*, me të cilin na njofton se variabla **n** është e padeklaruar.

Shembull

```
// Programi undef
#include <iostream>
using namespace std;
#define tekst "Koha e bukur"
int main()
{
    cout << "Teksti i parë: "
         << tekst;
#undef tekst
#define tekst "Koha me borë"
    cout << "\nTeksti i dytë: "
         << tekst
         << endl;
return 0;
}
```

Pas paraprocesimit, komanda e parë për shtypje është:

```
    cout << "Teksti i parë: "
         << "Koha e bukur";
```

kurse komanda e dytë për shtypje duket kështu:

```
    cout << "\nTeksti i dytë: "
         << "Koha me borë"
         << endl;
```

Direktivat për kompilim të kushtëzuar të pjesëve të programit

Këto direktiva, p.sh. mund të përdoret kur ekzistohen disa versione të një programi, gjë që shfrytëzohet nga shtëpitë softverike.

Në këtë grup komandash bëjnë pjesë komandat:

```
#if
#else
#elif
#endif
```

Përmes këtyre komandave, kompilohen vetëm pjesë të caktuara të programit, të cilat e plotësojnë kushtin e dhënë.

Shembull

```
// Programi if
#include <iostream>
using namespace std;
#define n 5
int main()
{
     int m=4;

#if n>3
          m=n+2;
#endif
     cout <<"\nVlera: "
          << m
          << "\n";
return 0;
}
```

Pas paraprocesimit, përkatësisht para kompilimit programi duket kështu:

```
.................
int main()
{
     int m=4;
     m=5+2;
     cout <<"\nVlera: " << m << "\n";
return 0;
}
```

Shembull

```
// Programi if
```

```
#include <iostream>
using namespace std;
#define n 5
int main()
{
    int m=4;

#if n>7
        m=n+2;
#endif
    cout <<"\nVlera: " << m << "\n";
return 0;
}
```

Pas paraprocesimit, përkatësisht para kompilimit programi duket kështu:

```
.................
int main()
{
    int m=4;
    cout <<"\nVlera: " << m << "\n";
return 0;
}
```

Në degë mund të përfshihen edhe më shum komanda..

Shembull

```
// Programi if
#include <iostream>
using namespace std;
#define n 5
int main()
{
    int m,y;
#if n>3
    {
        m=n+2;
        y=3*n+m-1;
        cout << "\nVlera e llogaritur: "
             << y << "\n";
    }
#endif
    cout <<"\nVlera: " << m << "\n";
return 0;
}
```

Shembull

```
// Programi if-else
#include <iostream>
using namespace std;
#define n 2
int main()
{
    int m=4;

#if (m+2)<10   // Nuk bën, në shprehje lejohen vetëm konstante!!!
        m=n+5;
#else
        cout << "\nTekst";
#endif
    cout <<"\nVlera: " << m << "\n";
return 0;
}
```

Shembull

```
// Programi if
#include <iostream>
using namespace std;
#define n 5
int main()
{
    cout << "Fillimi\n";
#if n==5
#include "C:\alfa\KlasaKoha.txt"
    koha dita;
    cout << "\nVlera per a: ";
    cin >> dita.a;
    cout << "\nVlera per b: ";
    cin >> dita.b;
#endif
    cout << "\nFundi" << "\n";
return 0;
}
```

Ku fajlli **KlasaKoha.txt** gjendet në **C:\alfa** i shkruar në Notepad dhe përmbajtja e tij është:

```
class koha
{
public:
     int a;
```

```
        double b;
    };
```

Pas paraprocesimit, përkatësisht para kompilimit, meqë plotësohet kushti te komanda **#if**, programi duket kështu:

```
.................
int main()
{
    cout << "Fillimi\n";
class koha
{
public:
    int a;
    double b;
};
    koha dita;
    cout << "\nVlera per a: ";
    cin >> dita.a;
    cout << "\nVlera per b: ";
    cin >> dita.b;
    cout << "\nFundi" << "\n";
return 0;
}
```

Versioni kur nuk plotësohet kushti, nëse p.sh. merret:

```
#define n 6
```

Pas paraprocesimit, përkatësisht para kompilimit, meqë nuk plotësohet kushti te komanda **#if**, programi duket kështu:

```
.................
int main()
{
    cout << "Fillimi\n";
    cout << "\nFundi" << "\n";
return 0;
}
```

Direktiva #elif

Nëse ka më shumë degëzime pas komandës **#if**, në vend të **#else** përdoret **#elif**.

Shembull

```
// Programi if
#include <iostream>
using namespace std;
#define alfa 3
#define beta 5
int main()
{
    int k;
#if alfa>4
      k=2*alfa+beta;
   #elif beta<7
      k=alfa+3*beta+1;
#endif
    cout << "Rezultati: "
         << k << endl;
return 0;
}
```

Pas paraprocesimit, përkatësisht para kompilimit, meqë nuk plotësohet kushti te komanda e parë **#if** por te komanda **#elif**, programi duket kështu:

```
.................
int main()
{
    int k;
    k=3+3*5+1;
    cout << "Rezultati: "
         << k << endl;
return 0;
}
================================================
```

Përdorimi i komandave paraprocesorike është me rëndësi nëse shfrytëzohen para programit, ku në fakt nuk kemi mundësi t'i shfrytëzojmë komandat e gjuhës C++.

Shembull

```
// Programi #if65
#include <iostream>
using namespace std;
#define dimensioni 25
#if dimensioni==20
    #undef dimensioni
    #define dimensioni 40
#elif dimensioni>15
```

```
    #undef dimensioni
    #define dimensioni 10
#else
    #undef dimensioni
    #define dimensioni 30
#endif
int main()
{
     int A[dimensioni],i,s;
     for (i=0;i<dimensioni;i++)
         if (i%2==0) A[i]=2*i+1;
         else A[i]=7;
     s=0;
     for (i=0;i<dimensioni;i++)
         if (A[i]==7)
             s=s+A[i];
     cout << "\nShuma s: "
          << s
          << "\n\n";
return 0;
}
```

Direktivat #ifdef dhe #ifndef

Shembull

```
// Programi ifdef
#include <iostream>
using namespace std;
#define x 10
int main()
{
    int k=0;
#ifdef x
    k=5*x+1;
#endif
    cout << "Rezultati k="
         << k << endl;
return 0;
}
```

Meqë plotësohet kushti **`#ifdef`**, pas paraprocesimit, përkatësisht para kompilimit, programi duket kështu:

```
.................
int main()
```

```
{
     int k=0;
     k=5*10+1;
     cout << "Rezultati k="
          << k << endl;
return 0;
}
```

Mund ta ketë edhe degën tjetër.

Shembull

```
// Programi ifdef
#include <iostream>
using namespace std;
#define d *
int main()
{
     int a=4,z;
#ifdef d
     z=5 d a+1;
#else
     z=a+3;
#endif
     cout << "Vlera e llogaritur z="
          << z << endl;
return 0;
}
```

Meqë plotësohet kushti **#ifdef**, pas paraprocesimit, përkatësisht para kompilimit, programi duket kështu:

```
.................
int main()
{
     int a=4,z;
     z=5 * a+1;
     cout << "Vlera e llogaritur z="
          << z << endl;
return 0;
}
```

Këtu të jepet shembull i formës:

```
#ifdef m
     int A[m];
```

pjesa:

```
    int A[m];
```

kompilohet vetem nese është definuar **m**.

Vlenë e kundërta:

```
#ifndef m
     #define m 10
#endif
     int A[m];
```

pjesa:

```
#define m 10
```

kompilohet vetëm nëse nuk është definuar **m**.

Direktiva #line

Përmes kësaj direktive programuesi i përcakton numrat rendor të rreshtave në program me qëllim të zbulimit të rreshtit në të cilin ka ndodhur gabim.

Shembull

```
// Programi line
#include <iostream>
using namespace std;
int main()
{
     double x,y;
     cout << "Vlera për x: ";
#line 50
     cin >> x;
     if (x<0)
          y=2*x+1;
     else
          y=x+3;
     cout << "Vlera e llogaritur y="
          << y << endl;
return 0;
}
```

Këtu, komandës:

```
cin >> x;
```

i është shoqëruar numri rendor **50**. Pas kësaj, nëse në rreshtat vijues ndodhë ndonjë gabim, në mesazhin e gabimit gjatë kompilimit të tij paraqitet numri duke e marrë si fillestar numrin e zgjedhur 50. P.sh. nëse shprehja:

```
y=x+3;
```

gabimisht shkruhet kështu:

```
y=a+3;
```

Në mesazhin e gabimit do të shënohet se gabimi ka ndodhur në rreshtin e 54, duke i pasur parasyshë numrimet vijuese:

```
50    cin >> x;
51    if (x<0)
52        y=2*x+1;
53    else
54        y=x+3;
```

Këto numra rendor shfrytëzohen edhe për vërejtjet që lidhen me rreshtat e veçantë të programit.

Brenda programit mund të vendosen edhe disa direktiva **#line** dhe te secila prej tyre të vendoset një numër rendor. Kjo mund të shfrytëzohet p.sh. kur programi është më i gjatë dhe numrat rendor fillohen për tanësi të veçanta të tij.

Direktiva #error

Kjo direktivë shfrytëzohet për gjenerimin e mesazhit i cili shkruhet në vazhdim të saj. Në këtë mënyrë mundësohet ndjekja e procesit të kompilimit deri në pozitën ku është vendosur direktiva, gjë që ka rëndësi kur kemi të bëjmë me programe më të gjata.

Shembull

```
// Programi error
#include <iostream>
using namespace std;
int main()
{
```

```
        double x,y;
        cout << "Vlera për x: ";
#error Pozita ku kontrollohet
        cin >> x;
        if (x<0)
            y=2*x+1;
        else
            y=a+3;
        cout << "Vlera e llogaritur y="
             << y << endl;
return 0;
}
```

Kur kompileri e takon direktivën në fjalë, e gjeneron mesazhin **`Pozita ku kontrollohet`** në ekran dhe e ndërpretë kompilimin e mëtejshëm. Nëse shfrytëzohet direktiva **`#line`**, ajo ka ndikim në numrin rendor të cilin e jepë si informatë të gabimit të gjeneruar edhe nga direktiva **`#error`**.

Direktiva #pragma

Kjo direktivë mundëson dhënjen e instrukcioneve të ndryshme për kompilerin. Në formë të përgjithshme kjo direktivë shkruhet kështu:

```
#pragma emri
```

Përmes emrit të shkruar në vazhdim të direktivës përcaktohet se cilat veçori (ang. features) specifike të makinës ose kompilerit aktivizohen. Meqë kjo direktivë lidhet me kompilerin, para shfrytëzimit të saj duhet të lexohet manueli i kompilerit përkatës.

---------- Vazhdimi 8.12.2009 ----------

Operatorët paraprocesorik

Në gjuhën C++ përdoren **3** operator paraprocesorik: **`defined`**, **`#`** dhe **`##`**.

Operatori defined

Ky operator shfrytëzohet për të testuar se a është definuar ose jo një identifikator.

Shembull

```
// Programi defined1
#include <iostream>
using namespace std;
#define pi 3.1515926
int main()
{
    double s,p,r=7;
#if defined(pi)
    {
        s=pi*r*r;
        cout << "Sipërfaqja e rrethit: "
             << s
             << "\n";
        p=2*pi*r;
        cout << "Perimetri i rrethit: "
             << p
             << "\n";
    }
#endif
     cout << "Fundi" << "\n\n";
return 0;
}
```

Rez.

Shtypen vlerat e llogaritura të sipërfaqes dhe perimetrit.

Nëse nuk është definuar identifikatori **pi**, do të shtypet vetëm fjala **Fundi**, kjo sepse nuk hyet në llogaritje.

Degëzimit mund t'i shoqërohet edhe dega **#else**.

Shembull

```
// Programi definedxx2
#include <iostream>
using namespace std;
#define pi 3.1515926
int main()
{
#if defined(pi)
    {
        double s,p,r=7;
        s=pi*r*r;
```

```
        cout << "Sipërfaqja e rrethit: "
             << s
             << "\n";
        p=2*pi*r;
        cout << "Perimetri i rrethit: "
             << p
             << "\n";
    }
#else
    {
        cout << "Nuk është definuar pi\n";
// Këtu mund të përfshihen shum komanda
    }
#endif
     cout << "Fundi" << "\n\n";
return 0;
}
```

Në këtë rast duhet pasur kujdes që deklarimi:

```
        double s,p,r=7;
```

të shënohet brenda kllapave dhe jo para komandës **#if**, sepse nëse nuk plotësohet kushti i dhënë, kompjuteri do ta kompiloj vetëm pjesën pas **#else**, duke na njoftuar njëkohësisht se variablave **s** dhe **p** nuk u referohemi askund.

Efektet e operatorit defined mund të mbërrihen edhe përmes komandave paraprocesorike **#ifdef**, ose **#ifndef**.

Shembull

```
// Programi defined1
#include <iostream>
using namespace std;
#define pi 3.1515926
int main()
{
    double s,p,r=7;
#ifdef pi
    {
        s=pi*r*r;
        cout << "Sipërfaqja e rrethit: "
             << s
             << "\n";
        p=2*pi*r;
        cout << "Perimetri i rrethit: "
             << p
             << "\n";
```

```
    }
#endif
     cout << "Fundi" << "\n\n";
return 0;
}
```

Operatori

Shembull

```
// Programi #
#include <iostream>
using namespace std;
#define alfa(x) #x
int main()
{
     cout << alfa(Jeta) << "\n";
return 0;
}
```

Rez. Jeta

Sikurse:

```
     cout << "Jeta" << "\n";
```

Shembull

```
// Operatori #
#include <iostream>
using namespace std;
#define shtypja(x) "Koha e " #x
int main()
{
    cout << shtypja(bukur);
         << "\n";
    return 0;
}
```

Këtu, gjatë ekzekutimit të komandës **cout**, përmes thirrjes së funkcionit **shtypja(bukur)**, në ekran do të shtypet teksti **Koha e bukur**. Kjo është rezultat i zëvendësimit të argumentit **x** me fjalën **bukur** me thonjëza, përkatësisht komanda **cout** që ekzekutohet është kështu:

```
     cout << "Koha e ""bukur";
```

ose:

```
cout << "Koha e bukur";
```

Shembull

Version i shembullit paraprak.

```
// Programi #
#include <iostream>
using namespace std;
#define shtypja(x) cout << "Koha e " #x << "\n"
int main()
{
     shtypja(bukur);
return 0;
}
```

Këtu kemi:

```
cout << "Koha e " "bukur" << "\n";
```

ose

```
cout << "Koha e bukur" << "\n";
```

Operatori

Ky operator përdoret për bashkimin e dy pjesëve.

```
// Programi ##
#include <iostream>
using namespace std;
#define bashkimi(x,y) x ## y
int main()
{
     cout << bashkimi("Koha e ","bukur") << "\n";
return 0;
}
```

Rez. Koha e bukur

Makro të paradefinuara

Në gjuhën C++ mund të shfrytëzohen 5 makro të paradefinuara.

Makroja __LINE__

Përmes saj jepet një numër i plotë i cili e paraqet numrin rendor të rreshtit aktual në program.

Shembull

```
// Programi __LINE__
#include <iostream>
using namespace std;
int main()
{
     const int m=8;
     int i,s;
     int A[m]={5,-2,7,4,1,8,10,6};
     s=0;
     for (i=0;i<m;i++)
          if (A[i]>3)
                if (A[i]<9)
                   s=s+A[i];
    cout << "Rreshti numër: "
         << __LINE__;
     cout << "\nShuma s="
             << s
             << "\n";
     return 0;
}
```

Do të shtypet numri rendor i rreshtit ku paraqitet makroja __LINE__.

Makroja __FILE__

Përmes saj merret një string i cili e paraqet emrin e fajllit që kompilohet.

Shembull

```
// Programi __FILE__
#include <iostream>
using namespace std;
int main()
{
    cout << "\nFajlli që kompilohet: "
         << __FILE__;
     const int m=8;
     int i,s;
     int A[m]={5,-2,7,4,1,8,10,6};
     s=0;
     for (i=0;i<m;i++)
```

```
          if (A[i]>3)
                if (A[i]<9)
                   s=s+A[i];
     cout << "\nShuma s="
          << s
          << "\n";
     return 0;
}
```

Makrot __DATE__ dhe __TIME__

Përmes tyre merret informata mbi datën dhe kohën kur ka filluar kompilimi i programit.

Shembull

```
// Programi __DATE__ dhe __TIME__
#include <iostream>
using namespace std;
int main()
{
    cout << "\nData e fillimit të kompilimit: "
         << __DATE__;
    cout << "\nKoha e fillimit të kompilimit: "
         << __TIME__;
     const int m=8;
     int i,s;
     int A[m]={5,-2,7,4,1,8,10,6};
     s=0;
     for (i=0;i<m;i++)
          if (A[i]>3)
                if (A[i]<9)
                   s=s+A[i];
     cout << "\nShuma s="
          << s
          << "\n";
     return 0;
}
```

Makroja __cplusplus

Përmes saj merret një numër i cili e identifikon kompilerin e gjuhës C++. Nëse kompileri është i pajtueshëm me standardet e gjuhës C++,
ai numër do të jetë më i madhë ose barazi me numrin **199711**, gjë që varet nga versioni i kompilerit.