3-Ma'ruza

3-Mavzu. Qurilish va arxitektura soxasida axborot jarayonlarini algoritmlash va dasturlash

Reja:

1. **If shartli operator;**
2. **Switch tanlash operatori;**
3. **Sikl jarayonlari;**
4. **For takrorlash operatori;**

Shartli operator. Shartli operator ikki ko`rinishda ishlatilishi mumkin:

Kengaytirilgan varianti	Qisqartirilgan varianti
If (ifoda) *1- operator;* *Else* *2- operator;*	If (ifoda) 1-operator;

Shartli operator bajarilganda avval ifoda hisoblanadi ; agar qiymat rost ya`ni noldan farqli bo`lsa 1- operator bajariladi. Agar qiymat yolg`on ya`ni nol bo`lsa va else ishlatilsa 2-operator bajariladi. Else qism har doim eng yaqin if ga mos qo`yiladi. Masalan,

```
if( n>0)
  if(a>b)
    Z=a;
   else
    Z=b;
```

Agar else qismni yuqori if ga mos qo`yish lozim bo`lsa, figurali qavslar ishlatish lozim.

```
if( n>0) {
  if(a>b)
    z=a;
}
else
    z=b;
```

Misol tariqasida uchta berilgan sonning eng kattasini aniqlash dasturini ko`ramiz:

6b-listing.	Output:

#include <iostream.h>
void main( )
{ float a,b,c,max);
cout <<"\n a="; cin>>a;
cout <<"\n b="; cin>>b;
cout <<"\n c="; cin>>c;
if (a>b)
if (a>c) max=a; else max=c;
else
if (b>c) then max=b; else max=c;
cout <<"\n" <<max; }
	max

Misol tariqasida kiritilgan ball va maksimal ball asosida baho aniqlanadi:

7-listing.	Output:

#include <iostream.h>
void main( )
{ float ball,max_ball,baho,d;
cout<< "\n ball="; cin>>ball;
cout<<"\n max_ball="; cin>>max_ball;
d=ball/max_ball;
	baho

```
 if (d>0.85) baho=5; else
   if (d>75) baho=4; else
   if (d>0.55) then baho=3; else baho=2;
  cout<<“\n” << baho;  }
```

Switch operatori. *if-else-if* yordami bilan bir necha shartni test qilishimiz mumkin. Lekin bunday yozuv nisbatan o`qishga qiyin va ko`rinishi qo`pol bo`ladi. Agar shart ifoda butun son tipida bo`lsa yoki bu tipga keltirilishi mumkin bo`lsa, biz switch (tanlash) ifodalarini ishlata olamiz. Switch ning umumiy ko`rinishi:

```
Switch(<ifoda>) {
    Case <1-qiymat>:<1-operator>
            …
     break;
            …
     default: <operator>
            …
     case: <n-operator>;       }
```

Oldin qavs ichidagi butun ifoda hisoblanadi va uning qiymati hamma variantlar bilan solishtiriladi. Biror variantga qiymat mos kelsa shu variantda ko`rsatilgan operator bajariladi. Agar biror variant mos kelmasa *default* orqali ko`rsatilgan operator bajariladi. *Break* operatori ishlatilmasa shartga mos kelgan variantdan tashqari keyingi variantdagi operatorlar ham avtomatik bajariladi. *Default; break* va belgilangan variantlar ixtiyoriy tartibda kelishi mumkin. *Default* yoki *break* operatorlarini ishlatish shart emas. Belgilangan operatorlar bo`sh bo`lishi ham mumkin.

Switch strukturasi bir necha *case* etiketlaridan (label) va majburiy bo`lmagan *default* etiketidan iboratdir. Etiket bu bir nomdir. U dasturnig bir nuqtasida qo`yiladi. Programmaning boshqa yeridan ushbu etiketga o`tishni bajarish mumkin. O`tish yoki sakrash goto bilan amalga oshiriladi, switch blokida ham qo`llaniladi.

5 lik sistemadagi bahoni so`zlik bahoga o`tqizadigan blokni yozaylik.

8-listing.	Output:

int baho = 4;
switch (baho) {
case 5: cout << "A`lo"; break;
case 4: cout << "Yaxshi"; break;
case 3: cout << "Qoniqarli"; break;
case 2:
case 1: cout << " Qoniqarsiz"; break;
default: cout << "Baho xato kiritildi!"; break; }
	Yaxshi

Switch ga kirgan o`zgaruvchi (yuqorigi misolda baho) har bir *case* etiketlarining qiymatlari bilan solishtirilib chiqiladi. Shartdagi qiymat etiketdagi qiymat bilan teng bo`lib chiqqanda ushbu *case* ga tegishli ifoda yoki ifodalar bloki bajariladi. So`ng *break* sakrash buyrug`i bilan *switch* ning tanasidan chiqiladi. Agar *break* qo`yilmasa, keyingi etiketlar qiymatlari bilan solishtirish bajarilmasdan ularga tegishli ifodalar ijro ko`raveradi. *default* etiketi majburiy emas. Lekin shart chegaradan tashqarida bo`lgan qiymatda ega bo`lgan hollarni tahlil qilish uchun kerak bo`ladi.

case va etiket orasida bo`sh joy qoldirish shartdir. Chunki, masalan, case 4: ni case4: deb yozish oddiy etiketni vujudga keltiradi, bunda sharti test qilinayotgan ifoda 4 bilan solishtirilmay o`tiladi.

Misol tariqasida bahoni son miqdoriga qarab aniqlash dasturini ko`ramiz.

9-listing.	Output:
Include <iostream.h> void main( ) { int baho; cin>> baho;	

```
switch(baho)
{case 2:cout <<“\n yomon”;break;
 case 3:cout <<“\n o`rta”;break;
 case 4:cout <<“\n yaxshi”;break;
 case 5:cout <<“\n a`lo”;break;
 default: cout <<“\n baho no`to`gri kiritilgan”; };}
```

Misol tariqasida kiritilgan simvol unli harf ekanligi aniqlanadi:

10-listing.

```
#Include <iostream.h>
void main( )
 {char c;
cin >> c;
switch(c)
{case `a`:
case `u`:
case `o`:
case `i`:
cout <<“\n Kiritilgan simvol unli harf” ;break;
default: cout <<“\n Kiritilgan simvol unli harf emas”;} ; }
```

Output:

While operatori. While operatori quyidagi umumiy ko`rinishga egadir:

```
While(ifoda)
  Operator
```

Bu operator bajarilganda avval ifoda hisoblanadi. Agar uning qiymati *false* dan farqli bo`lsa operator bajariladi va ifoda qayta hisoblanadi. To ifoda qiymati *false* bo`lmaguncha takrorlash qaytariladi.

Agar dasturda while (ture); satr qo`yilsa bu dastur hech qachon tugamaydi.

11-listing. **Berilgan n gacha sonlar yigindisi**

```
void main()
 { long n,i=1,s=0;
cin >>n;
while (i<= n )
  s+=i++;
cout<<” s=”<< s;    }
```

Output:

```
n=5;
s=15;
```

Bu dasturda *s+=i++* ifoda *s=s+i; i=i+1* ifodalarga ekvivalentdir.

Quyidagi dastur to nuqta bosilmaguncha kiritilgan simvollar va qatorlar soni hisoblanadi:

12-listing.

```
void main()
{ long nc=0, nl=0;
char c=``;
while (c!= `.` )
{++nc; cin >>c;
if (c ==`\n`) ++nl; };
cout<<("%1d\n", nc);
cout <<"\n satrlar="<< nl<<”simvollar=”<< nc; }
```

Output:

Do-While operatori. Do while *ifodasi* while *strukturasiga o`xshashdir. Bitta farqi shundaki* while *da shart boshiga tekshiriladi.* Do while *da esa takrorlanish tanasi eng kamida bir marta ijro ko`radi va shart strukturaning so`ngida test qilinadi. Shart* true *bo`lsa blok yana takrorlanadi. Shart* false *bo`lsa* do while *ifodasidan chiqiladi. Agar* do while *ichida qaytarilishi kerak bo`lgan ifoda bir dona bo`lsa {} qavslarning keragi yo`qdir. Quyidagicha bo`ladi:*

```
do
 ifoda;
while (shart);
```

Lekin {} qavslarning yo`qligi dasturchini adashtirishi mumkin. Chunki qavssiz *do while* oddiy *while* ning boshlanishiga o`xshaydi. Buni oldini olish uchun {} qavslarni har doim qo`yishni tavsiya etamiz.

```
int k = 1;
do {
  k = k * 5;
} while ( !(k>1000) );
```

Bu blokda 1000 dan kichik yoki teng bo`lgan eng katta 5 ga karrali son topilmoqda. *while* shartini ozroq o`zgartirib berdik, ! (not - inkor) operatorining ishlashini misolda ko`rsatish uchun. Agar oddiy qilib yozadigan bo`lsak, *while* shartining ko`rinishi bunday bo`lardi: *while (k<=1000);* Cheksiz takrorlanishni oldini olish uchun shart ifodasining ko`rinishiga katta e`tibor berish kerak. Bir nuqtaga kelib shart *true* dan *false* qiymatiga o`tishi shart.

13-listing**. Berilgan n gacha sonlar yigindisi**.	Output:

void main()
{ long n,i=1,s=0;
cin >>n;
do
s+=i++;
while (i<= n);
cout<<"\n s="<< s; }
	n=5;
	s=15;

Bu dasturning kamchiligi shundan iboratki agar n qiymati 0 ga teng yoki manfiy bo`lsa ham, takrorlash tanasi bir marta bajariladi va s qiymati birga teng bo`ladi.

Keyingi misolimizda simvolning kodini monitorga chiqaruvchi dasturni ko`ramiz. Bu misolda takrorlash to ESC (kodi 27) tugmasi bosilmaguncha davom etadi. Shu bilan birga ESC klavishasining kodi ham ekranga chiqariladi.

For operatori Richard L. Halterman Fundamentals of C++ Programming. Copyright © 2008–2017. All rights reserved. pg.129-131**.** For operatorining umumiy ko`rinishi quyidagicha:

```
For( 1-ifoda;2- ifoda; 3-ifoda)
Operator
```

Bu operator quyidagi operatorga mosdir.

```
1-ifoda;
while(2-ifoda) {
operator
3-ifoda }
```

15-listing**. Berilgan n gacha sonlar yigindisi.**	Output:

# include <iostream.h>; 	
void main {
int n;
cin>>n;
for(int i=1,s=0;i<=n; i++, s+=i);
cout<<"\n",s; }
	n=5;
	s=15;

FOR operatori tanasi bu misolda bo`sh, Lekin *C++* tili grammatikasi qoidalari *FOR* operatori tanaga ega bo`lishini talab qiladi. Bo`sh operatorga mos keluvchi nuqta vergul shu talabni bajarishga xizmat qiladi.

Keyingi dasturda kiritilgan jumlada satrlar, so`zlar va simvollar sonini hisoblanadi.

16-listing**.**	Output:

```
# include <iostream.h>;
#define yes 1
#define   no  0
void main()
{ int c, nl, nw, inword;
inword = no;
nl = nw = nc = 0;
for(char c=``;c!=`.`;cin>> c)
{++nc;
if (c == `\n`)
++nl;
if (c==` ` ||c==`\n` ||c==`\t`)
inword = no;
else if (inword == no)
inword = yes;
++nw; }
cout <<"\n satrlar="<< nl<<"so`zlar="<< nw<<"simvollar="<< nc; }
```

Programma har gal so`zning birinchi simvolini uchratganda, mos o`zgaruvchi qiyMatni bittaga oshiradi. *INWORD* o`zgaruvchisi programma so`z ichida ekanligini kuzatadi. Oldiniga bu o`zgaruvchiga so`z ichida emas ya`ni *NO* qiymati beriladi. *YES* va *NO* simvolik o`zgarmaslardan foydalanish dasturni o`qishni yengillashtiradi.

NL = NW = NC = 0 qatori quyidagi qatorga mos keladi: NC = (NL = (NW = 0));

For strukturasi sanovchi (*counter*) bilan bajariladigan takrorlashni bajaradi. Boshqa takrorlash bloklarida (*while*, *do/while*) takrorlash sonini *control* qilish uchun ham sanovchini qo`llasa bo`lardi, bu holda takrorlanish sonini o`ldindan bilsa bo`lardi, ham boshqa bir holatning vujudga kelish-kelmasligi orqali boshqarish mumkin edi. Ikkinchi holda ehtimol miqdori katta bo`ladi. Masalan, qo`llanuvchi belgilangan sonni kiritmaguncha takrorlashni bajarish kerak bo`lsa biz *while* li ifodalarni ishlatamiz. *for* da esa sanovchi ifodaning qiymati oshirilib (kamaytirilib) bosilaveradi, va chegaraviy qiymatni olganda takrorlanish tugatiladi. *for* ifodasidan keyingi bitta ifoda qaytariladi. Agar bir necha ifoda takrorlanishi kerak bo`lsa, ifodalar bloki *{}* qavs ichiga olinadi.

17-listing.	Output:

# include <iostream.h> 	
int main()
{ for (int i = 0; i == 5; i++) {
cout << i << endl; }
return (0);
}
	0
	1
	2
	3
	4
	5

for strukturasi uch qismdan iboratdir. Ular nuqtavergul [;] bilan bir-biridan ajratiladi. for ning ko`rinishi:

```
for( a; b; c ){
takror etiladigan blok }
```

a - e`lon va initsalizatsiya.

b - shartni tekshirish (oz`garuvchini chegaraviy qiymat bilan solishtirish).

c - o`zgaruvchining qiyMatni o`zgartirish.

Qismlarning bajarilish ketma-ketligi quyidagichadir:

Boshida a bajariladi (faqat bir marta), keyin b dagi shart tekshiriladi va agar u true *bo`lsa takrorlanish bloki ijro ko`radi, va eng oxirda c da o`zgaruvchilar o`zgartiriladi, keyin yana ikkinchi qismga o`tiladi. for strukturamizni* while *struktura bilan almashtirib ko`raylik:*

```
for (int i = 0; i < 10 ; i++)
 cout << "Hello!"<< endl;
```

Ekranga 10 marta Hello! so`zi bosib chiqariladi. i o`zgaruvchisi 0 dan 9 gacha o`zgaradi. i=10 bo`lganda esa *i < 10* sharti noto`g`ri (*false*) bo`lib chiqadi va *for* strukturasi nihoyasiga yetadi. Buni while bilan yozsak:

```
int i = 0;
while ( i<10 ){
 cout << "Hello!" << endl;
 i++; }
```

Endi for ni tashkil etuvchi uchta qismninig har birini alohida ko`rib chiqsak. Birinchi qismda asosan takrorlashni boshqaradigan sanovchi (counter) o`zgaruvchilar e`lon qilinadi va ularga boshlangich qiymatlar beriladi (initsalizatsiya). Yuqoridagi dastur misolida buni *int i = 0;* deb berganmiz. Ushbu qismda bir necha o`zgaruvchilarni e`lon qilishimiz mumkin, ular vergul bilan ajratilinadi. Ayni shu kabi uchinchi qismda ham bir nechta o`zgaruvchilarning qiyMatni o`zgartirishimiz mumkin. Undan tashqari birinchi qismda for dan oldin e`lon qilingan o`zgaruvchilarni qo`llasak bo`ladi. Masalan,

```
int k = 10;
int l;
for (int m = 2, l = 0 ; k <= 30 ; k++, l++, ++m) {
cout << k + m + l; }
```

Albatta bu ancha sun`iy misol, lekin u bizga *for* ifodasining naqadar moslashuvchanligini ko`rsatadi. *for* ning qismlari tushurib qoldirilishi mumkin. Masalan, *for(;;) {}* ifodasi cheksiz marta qaytariladi. Bu for dan chiqish uchun *break* operatorini beramiz. Yoki agar sanovchi sonni takrorlanish bloki ichida o`zgartirsak, *for* ning 3-qismi kerak emas. Masalan,

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

Yana qo`shimcha misollar beraylik.

```
for (int y = 100; y >= 0; y-=5){
...
ifoda(lar);
... }
```

Bu yerda 100 dan 0 gacha 5 lik qadam bilan tushiladi.

```
for(int d = -30; d<=30; d++){
...
ifoda(lar);
... }
```

60 marta qaytariladi.

For strukrurasi bilan dasturlarimizda yanada yaqinroq tanishamiz. Endi a e`lon qilinadigan o`zgaruvchilarning xususiyati haqida bir og`iz aytib o`taylik. Standartga ko`ra bu qismda e`lon qilingan o`zgaruvchilarning qo`llanilish sohasi faqat o`sha for strukturasi bilan chegaralanadi. Yani bitta blokda joylashgan for strukturalari mavjud bo`lsa, ular ayni ismli o`zgaruvchilarni qo`llana ololmaydilar. Masalan, quyidagi xatodir:

```
for(int j = 0; j<20; j++){...}
...
for(int j = 1; j<10 ; j++){...} //xato!
```

j o`zgaruvchisi birinchi for da e`lon qilib bo`lindi. Ikkinchi for da ishlatish mumkin emas. Bu masalani yechish uchun ikki xil yo`l tutish mumkin.

Birinchisi bitta blokda berilgan for larning har birida farqli o`zgaruvchilarni qo`llashdir. Ikkinchi yo`l for lar guruhidan oldin sanovchi vazifasini bajaruvchi bir o`zgaruvchini e`lon qilishdir. Va for larda bu o`zgaruvchiga faqat kerakli boshlangich qiymat beriladi xalos.

for ning ko`rinishlaridan biri, bo`sh tanali for dir.

```
for(int i = 0 ; i < 1000 ; i++);
```

Buning yordamida biz dastur ishlashini sekinlashtirishimiz mumkin.

Break operatori. Ba`zi hollarda takrorlash bajarilishini ixtiyoriy joyda to`xtatishga to`g`ri keladi. Bu vazifani break operatori bajarishga imkon beradi. Bu operator darhol takrorlash bajarilishini to`xtatadi va boshqaruvni takrorlashdan keyingi operatorlarga uzatadi. Masalan, o`quvchining n ta olgan baholariga qarab uning o`qish sifatini aniqlovchi dasturini ko`ramiz. Buning uchun dasturda o`quvchining olgan minimal bahosi aniqlanadi

18-listing.	Output:

```
# include <iostream.h>
void main()
{ int i,n,min,p;
while (1)
 { cout<<("Xato! n>0 bo`lishi lozim ! \n");
 cout<<"Baholar soni="; cin>>n;
if (n>0) break; };
for (i=1,min=5; i<=n; i++) {
  cin >> p;
  if ((p<2)||(p>5)) { min=0; break; };
 if (min>p) min=p;
if ((p<2)||(p>5)) break;
switch(min) {
case 0:cout<<"Baho noto`g`ri kiritilgan"; break;
case 2:cout<<"Talaba yomon o`qiydi";break;
case 3:cout<<"Talaba o`rtacha o`qiydi";break;
case 4:cout<<"alaba yaxshi o`qiydi";break;
case 5:cout<<"Talaba a`lo o`qiydi";break; }}}
```

Biz misolda xato kiritilgan n qiymatdan saqlanish uchun *while(1)* takrorlash kiritilgan. Agar *n>0* bo`lsa *Break* operatori takrorlashni to`xtatadi va dastur bajarilishi davom etadi. Agar kiritilayotgan baholar chegarada yotmasa min ga 0 qiymat berilib darhol takrorlashdan chiqiladi.

Continue operatori. Takrorlash bajarilishiga ta`sir o`tkazishga imkon beradigan yana bir operator *Continue* operatoridir. Bu operator takrorlash qadamini bajarilishini to`xtatib *for* va *while* da ko`rsatilgan shartli tekshirishga o`tkazadi.

Quyidagi misolda ketma-ket kiritilayotgan sonlarning faqat musbatlarining yig`indisini hisoblaydi. Sonlarni kiritish 0 soni kiritilguncha davom etadi.

19-listing.	Output:

```
# include <iostream.h>
void main()
{ int a,n=10,s=0;
 for (int i=1;i<=n;i++) {
 cin << a;
 if (a<=0) continue;
 s+=a;
 if (a=0) break; }
 cout << s; }
```

***O`tish operatori GO TO.*** *O`tish operatorining ko`rinishi:*

Go to <identifikator>

Bu operator identifikator bilan belgilangan operatorga o`tish kerakligini ko`rsatadi. Masalan, *goto A1;…;A1:y=5;* Strukturali dasturlashda *Go to* operatoridan foydalanmaslik maslahat beriladi. Lekin ba`zi hollarda o`tish operatoridan foydalanish dasturlashni

osonlashtiradi. Masalan, bir necha takrorlashdan birdan chiqish kerak bo`lib qolganda, to`g`ridan-to`g`ri *break* operatorini qo`llab bo`lmaydi, chunki u faqat eng ichki takrorlashdan chiqishga imkon beradi.

Quyidagi misolda n ta qatorga n tadan musbat son kiritiladi. Agar n yoki sonlardan biri manfiy bo`lsa, kiritish qaytariladi:

20-listing.	Output:

```
  # include <iostream.h>
  void main() {
int n,i,j,k;
M1: cout<<"\n n="; cin>>n;
if (n<=0) { cout<<"\n xato! n>0 bo`lishi kerak";
goto M1;} ;
  M: cout<<"x sonlarni kiriting \n";
for (i=1; i<=n; i++) {
cout<<"\n"<< i<<"=" ;
 cin>> k;
if (k<=0) goto M; }   }
```

Bu masalani *GOTO* operatorisiz hal qilish uchun qo`shimcha o`zgaruvchi kiritish lozimdir.

21-listing.	Output:

```
  # include <iostream.h>
  void main() {
int n, I, j, k;
   while(1) {
cout<<"\n n="; cin>>n;
  if (n>0) break;
cout<<"\n xato! n>0 bo`lishi kerak"; } ;
   int M=0;
  while (M)
  { M=0;
   cout<<"x sonlarni kiriting \n";
for (I=1; I<=10; I++) {
  if (M) break;
  cout<<("\n I=%", I);
  for (j=1 ; j<=10; j++) {
  cin>>k;
  if (k<=0)   {
  M=1; break;   }   }   }  } }
```

C++ da dasturlashning asosiy bloklaridan biri funksiyalardir. Funksiyalarning foydasi shundaki, katta masala bir necha kichik bo`laklarga bo`linib, har biriga alohida funksiya yozilganda, masala yechish algoritmi ancha soddalashadi. Bunda dasturchi yozgan funksiyalar *C++* ning standart kutubxonasi va boshqa firmalar yozgan kutubxonalar ichidagi funksiyalar bilan birlashtiriladi. Bu esa ishni osonlashtiradi. Ko`p holda dasturda takroran bajariladigan amalni funksiya sifatida yozish va kerakli joyda ushbu funksiyani chaqirish mumkin. Funksiyani programma tanasida ishlatish uchun u chaqiriladi, yani uning ismi yoziladi va unga kerakli argumentlar beriladi. () qavslar ushbu funksiya chaqirig`ini ifodalaydi. Masalan,

```
foo();
k = square(l);
```

Demak, agar funksiya argumentlar olsa, ular () qavs ichida yoziladi. Argumentsiz funksiyadan keyin esa () qavslarning o`zi qo`yiladi.

Funksiyalar dasturchi ishini juda yengillashtiradi. Funksiyalar yordamida programma modullashadi, qismlarga bo`linadi. Bu esa keyinchalik dasturni rivojlantirishni osonlashtiradi. Dastur yozilish davrida xatolarni topishni yengillashtiradi. Bir misolda funksiyaning asosiy qismlarini ko`rib chiqaylik.

```
int foo(int k, int t) {
  int result;
  result = k * t;
  return (result);}
```

Yuqoridagi *foo* funksiyamizning ismi, () qavslar ichidagi parametrlar – int tipidagi k va t lar kirish argumentlaridir, ular faqat ushbu funksiya ichida ko`rinadi va qo`llaniladi. Bunday o`zgaruvchilar *lokal*(local-mahalliy) deyiladi. *result foo()* ning ichida e`lon qilinganligi uchun u ham lokaldir. Demak biz funksiya ichida o`zgaruvchilarni va sinflarni (class) e`lon qilishimiz mumkin ekan. Lekin funksiya ichida boshqa funksiyani e`lon qilib bo`lmaydi. foo() funksiyamiz qiymat ham qaytaradi. Qaytish qiyMatning tipi foo() ning e`lonida eng boshida kelgan - int tipiga ega. Biz funksiyadan qaytarmoqchi bo`lgan qiymatning tipi ham funksiya e`lon qilgan qaytish qiymati tipiga mos kelishi kerak - ayni o`sha tipda bo`lishi yoki o`sha tipga keltirilishi mumkin bo`lgan tipga ega bo`lishi shart. Funksiyadan qiymatni return ifodasi bilan qaytaramiz. Agar funksiya hech narsa qaytarmasa e`londa void tipini yozamiz. Yani:

```
void funk(){
  int g = 10;
  cout << g;
  return;}
```

Bu funksiya void (bo`sh, hech narsasiz) tipidagi qiymatni qaytaradi. Boshqacha qilib aytganda qaytargan qiymati bo`sh to`plamdir. Lekin funksiya hech narsa qaytarmaydi deya olmaymiz. Chunki hech narsa qaytarmaydigan maxsus funksiyalar ham bor. Ularning qaytish qiymati belgilanadigan joyga hech narsa yozilmaydi. Biz unday funksiyalarni keyinroq ko`rib chiqamiz. Bu yerda bir nuqta shuki, agar funksiya maxsus bo`lmasa, Lekin oldida qaytish qiymati tipi ko`rsatilmagan bo`lsa, qaytish qiymati int tipiga ega deb qabul qilinadi.

Void qaytish tipli funksiyalardan chiqish uchun *return;* deb yozsak yetarlidir.Yoki return ni qoldirib ketsak ham bo`ladi. Funksiyaning qismlari bajaradigan vazifasiga ko`ra turlicha nomlanadi. Yuqorida ko`rib chiqqanimiz funksiya aniqlanishi (function definition) deyiladi, chunki biz bunda funksiyaning bajaradigan amallarini funksiya nomidan keyin,{} qavslar ichida aniqlab yozib chiqyapmiz. Funksiya aniqlanishida {} qavslardan oldin nuqta-vergul [;] qo`yish xatodir. Bundan tashqari funksiya e`loni, prototipi yoki deklaratsiyasi (*function prototype*) tushunchasi qo`llaniladi. Bunda funksiyaning nomidan keyin hamon nuqta-vergul qo`yiladi, funksiya tanasi esa berilmaydi. *C++* da funksiya qo`llanilishidan oldin uning aniqlanishi yoki hech bo`lmaganda e`loni kompilyatorga uchragan bo`lishi kerak. Agar funksiya e`loni boshqa funksiyalar aniqlanishidan tashqarida berilgan bo`lsa, uning kuchi ushbu fayl oxirigacha boradi. Biror bir funksiya ichida berilgan bo`lsa kuchi faqat o`sha funksiya ichida tarqaladi. E`lon fayllarda aynan shu funksiya e`lonlari berilgan bo`ladi. Funksiya e`loni va funksiya aniqlanishi bir-biriga mos tushishi kerak. Masalan,

```
double square(char, bool);
float average(int a, int b, int c);
```

Funksiya e`lonlarda kirish parametrlarining faqat tipini yozish kifoya, xuddi *square()* funksiyasidek. Yoki kiruvchi parametrlarning nomi ham berilishi mumkin, bu nomlar kompilyator tarafidan etiborga olinmaydi, biroq dasturning o`qilishini ancha osonlashtiradi. Bulardan tashqari *C++* da funksiya imzosi (*function signature*) tushunchasi bor. Funksiya imzosiga funksiya nomi, kiruvchi parametrlar tipi, soni, ketma-ketligi kiradi. Funksiyadan qaytuvchi qiymat tipi imzoga kirmaydi.

```
int foo();       //1
int foo(char, int);  //2
```

```
double foo();     //3 - 1 funksiya bilan imzolari ayni.
void foo(int, char);  //4 - 2 bilan imzolari farqli.
char foo(char, int);  //5 - 2 bilan imzolari ayni.
int foo(void);     //6 - 1 va 3 bilan imzolari ayni.
```

Yuqoridagi misolda kirish parametrlari bo`lmasa biz () qavsning ichiga void deb yozishimiz mumkin (6 ga qarang). Yoki () qavslarning quruq o`zini yozaversak ham bo`ladi (1 ga qarang). Yana bir tushuncha - funksiya chaqirig`idir. Dasturda funksiyani chaqirib, qo`llashimiz uchun uning chaqiriq ko`rinishini ishlatamiz. () qavslari funksiya chaqirig`ida qo`llaniladi. Agar funksiyaning kirish argumentlari bo`lmasa, () qavslar bo`sh holda qo`llaniladi. Aslida () qavslar *C++* da operatorlardir. Funksiya kirish parametrlarini har birini ayri-ayri yozish kerak, masalan, *float average(int a, int b, int c);* funksiyasini *float average(int a,b,c);* deb yozishimiz xatodir.

Hali aytib o`tganimizdek, funksiya kirish parametrlari ushbu funksiyaning lokal o`zgaruvchilaridir. Bu o`zgaruvchilarni funksiya tanasida boshqattan e`lon qilish sintaksis xatoga olib keladi.

27-listing.

```
# include <iostream.h>
int foo(int a, int b); //Funksiya prototipi,
        //argumentlar ismi shart emas.
 int main()
{  for (int k = 1; k<6; k++){
   for (int l = 5; l>0; l--){
     cout << foo(k,l) << " ";    //Funksiya chaqirig`i.
   }//end for (l...)
   cout << endl;
  }//end for (k...)
return (0);
} //end main()
//foo() funksiyasining aniqlanishi
int foo(int c, int d)
{ //Funksiya tanasi
  return(c * d); }
```

Output:

```
5 4 3 2 1
10 8 6 4 2
15 12 9 6 3
20 16 12 8 4
25 20 15 10 5
```

Bizda ikki sikl ichida *foo()* funksiyamiz chaqirilmoqda. Funksiyaga k va l o`zgaruvchilarining nusxalari uzatilmoqda. Nushalarning qiymati mos ravishda funksiyaning aniqlanishida berilgan c va d o`zgaruvchilarga berilmoqda. k va l ning nushalari deganimizda adashmadik, chunki ushbu o`zgaruvchilarining qiymatlari funksiya chaqirig`idan hech qanday ta`sir ko`rmaydi. *C++* dagi funksiyalarning bir noqulay tarafi shundaki, funksiyadan faqat bitta qiymat qaytadi. Undan tashqari yuqorida ko`rganimizdek, funksiyaga berilgan o`zgaruvchilarning faqat nushalari bilan ish ko`rilarkan. Ularning qiyMatni normal sharoitda funksiya ichida o`zgartirish mumkin emas. Lekin bu muammolar ko`rsatkichlar yordamida osonlikcha hal etiladi. Funksiya chaqiriqlarida avtomatik ma`lumot tipining konversiyasi bajariladi. Bu amal kompilyator tomonidan bajarilganligi sababli funksiyalarni chaqirganda ehtiyot bo`lish kerak. Javob xato ham bo`lishi mumkin. Shu sababli kirish parametrlar tipi sifatida katta hajmli tiplarni qo`llash maqsadga muvofiq bo`ladi. Masalan, double tipi har qanday sonli tipdagi qiymatni o`z ichiga olishi mumkin. Lekin bunday qiladigan bo`lsak, biz tezlikdan yutqazishimiz turgan gap. Avtomatik konversiyaga misol keltiraylik.

28-listing.

```
int division(int m, int k) {
return (m / k);
}
dasturda chaqirsak:...
float f = 14.7;
```

Output:

```
4
```

```
double d = 3.6;
int j = division(f,d); //f 14 bo`lib kiradi, d 3 bo`lib kiradi
        // 14/3 - butun sonli bo`lish esa 4 javobini beradi
cout << j;
```

Demak, kompilyator f va d o`zgaruvchilarining kasr qismlarini tashlab yuborar ekan. Qiymatlarni pastroq sig`imli tiplarga o`zgartirish hatoga olib keladi.