3.4. MUTAXASSISLIK MASALALARINI DASTURLASHDA TAKRORLANUVCHI ALGORITMDAN FOYDALANISH

Ko'pincha masalalarni dasturlashda amallar ketma-ketligini bir necha bor bajarishga to`g`ri keladi. Bunday jarayon sikl deb ataladi. Berilgan masalani yechishda, unda qatnashgan o'zgaruvchi(parametr)ning bir qancha qiymatlarida bajariladigan amallar ketma-ketligi takrorlanuvchi algoritmlar deb ataladi.

Takrorlanuvchi algoritmlarda takrorlanishlar soni nechta bo'lishi mumkin?

Takrorlanuvchi algoritmlarda bajariladigan amallar soni yoki takrorlanishlar soni berilgan masalaga bog'liq ravishda bir nechta bo'lib, ularning soni aniq yoki noaniq bo'lishi mumkin.

3.4.1. Sikl operatorlari

***While* operatori**. While operatori quyidagi umumiy ko`rinishga ega:

While operatorining umumiy shakli quyidagicha:

```
while (ifoda)
{
  // operatorlar ketma-ketligi
  // ...
}
```

bu erda, ***ifoda*** - C++ tilidagi mantiqiy ifodadir. Operatorlar ketma-ketligi ***ifoda rost*** qiymatni qabul qilsa bajariladi. ***Ifoda yolg`on*** qiymatni qabul qilgandan so'ng, ***while*** operatorining bajarilishi to'xtaydi va boshqaruv ***while*** dan keyingi operatorga o'tadi.

Ya'ni, bu operator bajarilganda avval ifoda hisoblanadi. Agar uning qiymati 0 dan farqli bo`lsa operator bajariladi va ifoda qayta hisoblanadi. To ifoda qiymati 0 bo`lmaguncha sikl qaytariladi.

Agar dasturda *while* (1); satr quyilsa bu dastur hech qachon tugamaydi.

Sikldagi takrorlanishlar soni noma'lum bo`lib, ma'lum shartga bog'liq bo'lsa, siklni tashkil qilishda ***While*** operatoridan foydalanish mumkin.

Misol. Berilgan n gacha bo’lgan sonlar yigindisi.

```
Voidmain()
{
long n,i=1,s=0;
cin>>n;
while (i<= n )
s+=i++;
Cout<<”\n s=”<< s;
};
```

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

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

```
Void main()
{
long nc=0,nl=0;
char c=’’;
while (c!= ‘.’ )
{++nc;
if (c =='\n') ++nl;
};
Cout<<("%1d\n", nc);
Cout <<"\n satrlar="<< nl<<”simvollar=”<< nc;
};
```

Do-While operatori. *Do-While* operatori umumiy ko`rinishi qo`yidagicha:

```
do
operator
While(ifoda)
```

Sikl operatorining bu ko`rinishida avval operator bajariladi so`ngra ifoda hisoblanadi. Agar uning qiymati 0 dan farqli bo`lsa operator yana bajariladi va hokazo. To ifoda qiymati 0 bo`lmaguncha sikl qaytariladi.

Misol. Berilgan n gacha sonlar yigindisi.

```
Void main()
{
long n,i=1,s=0;
cin >>n;
do
s+=i++;
while (i<= n );
Cout<<”\n s=”<< s;
};
```

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

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

```
# include <iostream.h>;
main ()
{ char d; int I;
do
cin>>d;
i=c;
Cout<<“\n “<<i;
while(i!=27);
};
```

2-Misol. Haqiqiy a soni berilgan. Quyidagi shartni qanoatlantiruvchi eng kichik n ni toping. $\mathbf{1} + \frac{\mathbf{1}}{\mathbf{2}} + \frac{\mathbf{1}}{\mathbf{3}} + \cdots + \frac{\mathbf{1}}{\boldsymbol{n}} > a$

Berilgan shartni qanoatlantiruvchi eng kichik n ni topish uchun tuzilgan dastur:

```
# include <iostream>
using namespace std;
int main()     {
float a, sum;
int n;
// a va n ni  kiritish
a = 2.2;
n = 1;
sum = 1.0/n;
while (sum <= a)
{
n++;
   sum = sum + 1.0/n;
}
cout<<"\n sum="<<sum;
}
```

For operatori.

Agar algoritmda takrorlanishlar soni aniq bo’lsa, for operatori ishlatiladi. for operatorining umumiy ko’rinishi quyidagicha:

```
for (initsializatsiya; ifoda; o'sish)
{
   // operatorlar ketma-ketligi
   // ...
}
```

Bu yerda:

- *initsalizatsiya*- o`zlashtirish operatori bo`lib, unda sikl o`zgaruvchisining boshlang`ich qiymati beriladi. ushbu o`garuvchi sikl ishini bosharuvchi xisoblagich vazifasini bajaradi;
- *ifoda* - sikl o'zgaruvchisining qiymati tekshiriladigan shartli ifoda. Ushbu bosqichda siklning keyingi bajarilishi aniqlanadi;
- *o'sish* - har bir iteratsiyadan keyin sikl o'zgaruvchisining qiymati qanday o'zgarishini aniqlaydi.
- *For* operatori *ifoda rost* qiymatni qabul qilsa, bajarilaveradi. *Ifodaning* qiymati *yolg`on* bo'lgandan so'ng to'xtaydi va *for* dan keyingi operator bajariladi.

1-misol. 100 dan 300 gacha bo'lgan butun sonlarning yig'indisini toping.

```
# include <iostream>
using namespace std;
int main()
{
int sum,i;
sum = 0;
for(i = 100; i <= 300; i++)
sum = sum + i;
cout<<"summa="<<sum;
}
```

Yoki ***For*** operatori umumiy ko`rinishi qo`yidagicha:

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

Operator

Bu operator qo`yidagi operatorga mosdir.

1-ifoda;

while(2-ifoda)

{

operator

3-ifoda

```
}
```

Misol. Berilgan n gacha sonlar yigindisini toppish uchun dastur tuzing.

```
# include <iostream.h>;
void main
{
int n;
Cin>>n;
for(int i=1,s=0;i<=n; i++, s+=i);
Cout<<”\n”,s;
};
```

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.

```
# 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<<”suzlar=”<< nw<<”simvollar=”<< nc;
}
```

Dastur har gal so`zning birinchi simvolini uchratganda, mos o`zgaruvchi qiymatini bittaga oshiradi. *INWORD* o`zgaruvchisi dastur 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 engillashtiradi.

NL = NW = NC = 0 qatori quyidagi qatorga mos keladi;

NC = (NL = (NW = 0));

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* strukturasi bir necha case belgilaridan (*label*) va majburiy bo`lmagan *default* belgisidan iboratdir. Belgi bu bir nomdir. U dasturnig bir nuqtasidaga qo`yiladi. Dasturning boshqa eridan ushbu belgiga 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.

```
int baho;
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 << "a’lo";
break;
default: cout << "baho noto`g`ri kiritilgan!";
break;
}
```

switch ga kirgan o`zgaruvchi (yuqorigi misolda baho) har bir *case* belgilarining qiymatlari bilan solishtirilib chiqiladi. Solishtirish yuqoridan pastga bajariladi. Shartdagi qiymat belgidagi 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 belgilar qiymatlari bilan solishtirish bajarilmasdan ularga tegishli ifodalar ijro ko`raveradi. Bu albatta biz istamaydigan narsa.

default belgi majburiy emas. Lekin shart chegaradan tashqarida bo`lgan qiymatda ega bo`lgan hollarni diagnostika qilish uchun kerak bo`ladi. *case* va etiket orasida bo`sh joy qoldirish shartdir. Chunki, masalan, *case* 4: ni *case4*: deb yozish oddiy belgini vujudga keltiradi, bunda sharti test qilinayotgan ifoda 4 bilan solishtirilmay o`tiladi.

***Do while* takrorlash operatori**.

Bu operator takrorlanishlar soni ma’lum bo‘lmagan sikllarni tashkil etishda ishlatiladi. Bu operatorning ishlash jarayonida har bir qadamdan keyin shart tekshiriladi. **do...while** operatorida takrorlanishlar kamida bir marta bajariladi. **do…while** opratorining **for** va **while** operatorlaridan farqi shart sikl oxirida tekshiriladi.

do ... while operatorining umumiy shakli:

```
do
{
   // operatorlar ketma-ketligi
   // ...
}
```

while (ifoda);

Do while ifodasi *while* strukturasiga o`xshashdir. Bitta farqi shundaki *while* da shart boshida 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`q.

Masalan:

Do ifoda;

while (shart);

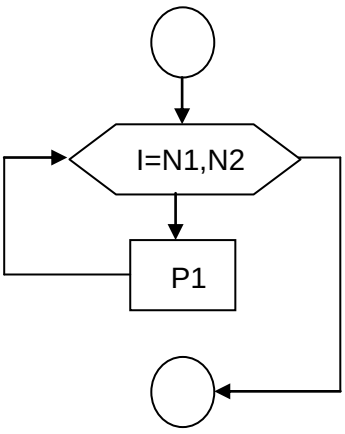
Lekin {} qavslarning yo`qligi dasturchini adashtirishi mumkin. Chunki qavssiz *do while* oddiy *while* ning boshlanishiga o`xshaydi. Buni oldini olish uchun {} qavslarni har doim qo`yishni tavsiya etamiz.

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

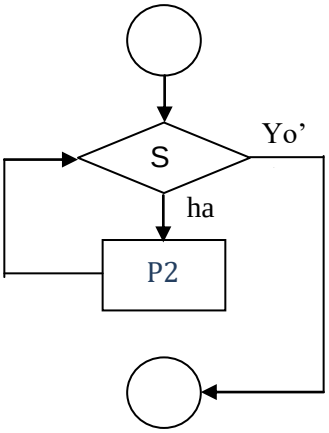
Bu blokda 1000 dan kichik yoki teng bo`lgan eng katta 5 ga karrali son topilmoqda. *while* shartini ozroq o`zgartirib berdik, ! (*not* - inkor) operatorining ishlashini misolda ko`rsatish uchun. Agar oddiy qilib yozadigan bo`lsak, *while* shartining ko`rinishi bunday bo`lardi: *while* (k<=1000); Cheksiz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

3-Misol. Argument ***x*** ***a*** dan ***b*** gacha ***h*** qadam bilan o‘zgarganda quyidagi funksiya qiymatlarini hisoblash dasturini ***do while*** operatoridan foydalanib tuzamiz. $Y = e^{\cos x} + \log_c (x+4)$

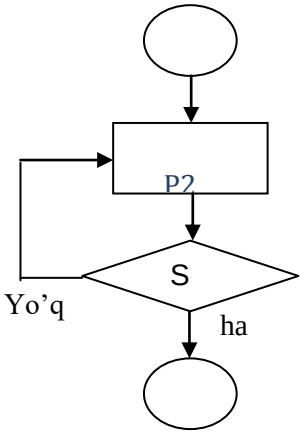
```
# include <iostream>
# include <math.h>
using namespace std;
int main()
{
float a,b,h,c,y,x;
cout<<"a,b,h,c larni kiriting \n"; cin>>a>>b>>h>>c;
x=a;
do
{
y=exp(cos(x))+log(x+4)/log(c);
cout<<" x="<<x<<"   y="<<y<<endl;
x+=h;
while (x<=b);
}
```



1-rasm



2-rasm



3-rasm

3.4.2. O`tish operatorlari

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

```
# include <iostream.h>
void main()
{
int I,n,min,p;
while (1)
{Cout<<“Baholar soni=”; Cin>>n;};
If (n>0) break;
Cout<<(“Xato! n>0 bo`lishi kerak! \n”);
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) cout 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<<” Talaba yaxshi o`qiydi”;break;
case 5:cout<<” Talaba a’lo o`qiydi”;break;
}
```

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

***Continue* operatori.** Sikl bajarilishiga ta’sir o`tkazishga imkon beradigan yana bir operator Continue operatoridir. Bu operator sikl qadamini bajarilishini tuxtatib *for* va *while* da ko`rsatilgan shartli tekshirishga o`tkazadi.

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

```
# include <iostream.h>
void main()
{
double s, x;
int x;
Cout<<(“\n 0 bilan tugallanuvchi sonlar qatorini kiriting \n”);
for (x=1.0; s=0.0; k=0; x!=0.0);
{
Cin>>(“%lf”, &x);
if (x<=0.0) continue;
k++; s+=x;
}
Cout<<“\n yig`indi=”<<s<<”musbat sonlar =”<<k;
}
```

3.4.3.Go to o`tish operatori.

O`tish operatorining ko`rinishi: **Goto <identifikator>**. Bu operator identifikator bilan belgilangan operatorga o`tish kerakligini ko`rsatadi. Misol uchun *goto* A1;…;A1: y=5; Strukturali dasturlashda *Go to* operatoridan foydalanmaslik maslahat beriladi. Lekin ba’zi hollarda o`tish operatoridan foydalanish dasturlashni osonlashtiradi. Misol uchun bir necha sikldan birdan

chiqish kerak bo`lib qolganda, to`g`ridan-to`g`ri *break* operatorini qo`llab bo`lmaydi, chunki u faqat eng ichki sikldan chiqishga imkon beradi.

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

```
# include <iostream.h>
int n, I, j, k;
M1: Cout<<“\n n=”; Cin>>n;
If (n<=0)
{
Cout<<“\n baho! n>0 bo`lishi kerak”;
Go to M1;
} ;
M: Cout<<“x sonlarni kiriting \n”;
For (I=1; I<=10; I++)
{
Cout<<“\n I=”<< i;
For (j=1 ; j<=10; j++)
{
Cin>> k;
if (k<=0) goto M;
}
}
```

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

```
# include <iostream.h>
int n, I, j, k;
while 1
{
Cout<<“\n n=”; Cin>>n;
```

```
if (n>0) break;
Cout<<“\n xato! n>0 bulishi 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>>(“%f”, k);
if (k<=0)
{
M=1;break;
} }
```

1-misol

```
#include <iostream>
 using namespace std;
 int main ()
{
 Double a = 1, b;
 Goto miss;
       b = 5 * a;
       miss: b = a + 1
  cout << “ b=’’ << b;
        return θ;
```

```
#include <iostream>
Using namescape std;
Int main () {
    double a= 1, b;
    nish: b= 5 * a;
        goto nish ;
         b = a + 1;
    cout << ‘’ b=” << b;
            return θ ;
}
```

} Bu dasturda goto operatori ishlagandan so’ng b=5 * a; operator ishlanmasdan tashlab ketiladi va ishlash navbati b=a+1; operatoriga berildi.	Dasturda qo’nish joyi uchish joyidan oldin ham yozilishi mumkin.

3.4.4. Qiymat berish operatorlari

Bu qismda keyingi bo`limlarda kerak bo`ladigan tushunchalarni berib o`tamiz.C++ da hisoblashni va undan keyin javobni o`zgaruvchiga beruvchi bir necha operator mavjuddir. Misol uchun: k = k * 4; ni k *= 4; deb yozsak bo`ladi. Bunda *= operatorining chap argumenti o`ng argumentga qo`shiladi va javob chap argumentda saqlanadi. Biz har bir operatorni ushbu qisqartirilgan ko`rinishda yoza olamiz (+=, -=, /=, *= %=). Ikkala qism birga yoziladi. Qisqartirilgan operatorlar tezroq yoziladi, tezroq kompilyasiya qilinadi va ba’zi bir hollarda tezroq ishlaydigan mashina kodi tuziladi. 1 ga oshirish va kamaytirish operatorlari (increment va decrement) C++ da bir argument oluvchi inkrenet (++) va dekrement (--) operatorlari mavjuddir. Bular ikki ko`rinishda ishlatilinadi, biri o`zgaruvchidan oldin (++f - preinkrement, --d - predekrement), boshqasi o`zgaruvchidan keyin (s++ -postinkrement, s-- - predekrement) ishlatilgan holi. Bularning bir-biridan farqini aytib o`taylik. Postinkrementda o`zgaruvchining qiymati ushbu o`zgaruvchi qatnashgan ifodada ishlatiladi va undan keyin qiymati birga oshiriladi. Preinkrementda esa o`zgaruvchining qiymati birga oshiriladi, va bu yangi qiymat ifodada qo`llaniladi. Predekrement va postdekrement ham aynan shunday ishlaydi lekin qiymat birga kamaytiriladi. Bu operatorlar faqatgina o`zgaruvchining qiymatini birga oshirish(kamaytirish) uchun ham ishlatilinishi mumkin, ya’ni boshqa ifoda ichida qo`llanilmasdan. Bu holda pre va post formalarining farqi yo`q.

Masalan:

++r;

r++;

Yuqoridagilarning funksional jihatdan hech qanday farqi yo`q, chunki bu ikki operator faqat r ning qiymatini oshirish uchun qo`llanilmoqda. Bu operatorlarni oddiy holda yozsak:

r = r + 1;

d = d - 1;

Lekin bizning inkrement/dekrement operatorlarimiz oddiygina qilib o`zgaruvchiga bir qo`shish(ayirish)dan ko`ra tezroq ishlaydi. Yuqoridagi operatorlarni qo`llagan holda bir dastur yozaylik.

```
// Postinkremet, preinkrement va qisqartirilgan teglashtirish operatorlari
# include <iostream.h>
int main()
{
int k = 5, l = 3, m = 8;
cout << k++ << endl; //ekranga 5 yozildi, k = 6 bo`ldi.
l += 4; // l = 7 bo`ldi.
cout << --m << endl; // m = 7 bo`ldi va ekranga 7 chiqdi.
m = k + (++l); // m = 6 + 8 = 14;
return (0);
}
```

Dasturdagi o`zgaruvchilar e’lon qilindi va boshlang`ich qiymatlarni olishdi. *cout << k++ << endl;* ifodasida ekranga oldin k ning boshlang`ich qiymati chiqarildi, keyin esa uning qiymati 1 da oshirildi. l += 4; da l ning qiymatiga 4 soni qo`shildi va yangi qiymat l da saqlandi. *cout << --m << endl;* ifodasida m ning qiymati oldin predekrement qilindi,va undan so`ng ekranga chiqarildi. m = k + (++l); da oldin l ning qiymati birga oshirildi va l ning yangi qiymati k ga qo`shildi. m esa bu yangi qiymatni oldi. Oshirish va kamaytirish operatorlari va ularning argumentlari orasida bo`shliq qoldirilmasligi kerak. Bu operatorlar sodda ko`rinishdagi o`zgaruvchilarga nisbatan qo`llanilishi mumkin xolos.

Masalan: ++(f * 5); ko`rinish noto`g`ridir.

3.4.5. Mantiqiy operatorlar

Boshqaruv strukturalarida shart qismi bor dedik. Shu paytgacha ishlatgan shartlarimiz ancha sodda edi. Agar bir necha shartni tekshirmoqchi bo`lganimizda ayri-ayri shart qismlarini yozardik. Lekin C++ da bir necha sodda shartni birlashtirib, bitta murakkab shart ifodasini tuzishga yordam beradigan mantiqiy operatorlar mavjuddir. Bular mantiqiy VA - && (and), mantiqiy YOKI - || (*or*) va mantiqiy INKOR - ! (not). Bular bilan misol keltiraylik. Faraz qilaylik, bir amalni bajarishdan oldin, ikkala shartimiz (ikkitadan ko`p ham bo`lishi mumkin) *true* (haqiqat) bo`lsin.

if (i < 10 && l >= 20){...}. Bu yerda {} qavslardagi ifodalar bloki faqat i 10 dan kichkina va l 20 dan katta yoki teng bo`lgandagina ijro ko`radi. *And* ning (&&) jadvali:

ifoda1	ifoda2	ifoda1 && ifoda2
false (0)	*false (0)*	*false (0)*
true (1)	*false (0)*	*false (0)*
false (0)	*true (1)*	*false (0)*
true (1)	*true (1)*	*true (1)*

Bu yerda *true* ni o`rniga 1, false ni qiymati o`rniga 0 ni qo`llashimiz mumkin.

Boshqa misol:

while (g<10 || f<4)

{

...

}

Bizda ikkita o`zgaruvchi bor (g va f). Birnchisi 10 dan kichkina yoki ikkinchisi 4 dan kichkina bo`lganda *while* ning tanasi takrorlanaveradi. Ya’ni shart ajarilishi uchun eng kamida bitta *true* bo`lishi kerak, *and* da (&&) esa hamma oddiy shartlar *true* bo`lishi kerak. *Or* ning (||) jadvali:

ifoda1	ifoda2	ifoda1 || ifoda2
false (0)	*false (0)*	*false (0)*
true (1)	*false (0)*	*true (1)*
false (0)	*true (1)*	*true (1)*
true (1)	*true (1)*	*true (1)*

&& va || operatorlari ikkita argument olishadi. Bulardan farqli o`laroq, !(mantiqiy inkor) operatori bitta argument oladi, va bu argumentidan oldin qo`yiladi. Inkor operatori ifodaning mantiqiy qiymatini teskarisiga o`zgartiradi. Ya'ni *false* ni *true* deb beradi, *true* ni esa *false* deydi.

Misol uchun:

```
if ( !(counter == finish) )
cout << student_bahosi << end;
```

Agar counter o`zgaruvchimiz *finish* ga teng bo`lsa, *true* bo`ladi,bu *true* qiymat esa ! yordamida *false* ga aylanadi. *false* qiymatni olgan *if* esa ifodasini bajarmaydi. Demak, ifoda bajarilishi uchun bizga *counter finish* ga teng bo`lmagan holati kerak. Bu yerda ! ga tegishli ifoda () qavslar ichida bo`lishi kerak.Chunki, mantiqiy operatorlar tenglilik operatorlaridan kuchliroqdir. Ko`p hollarda ! operatori o`rniga mos keladigan mantiqiy tenglilik yoki solishtirish operatorlarini ishlatsa bo`ladi, masalan yuqoridagi misol quyidagi ko`rinishda bo`ladi:

```
if (counter != finish)
cout << student_bahosi << endl;
```

Not ning jadvali:

ifoda	!(ifoda)
false (0)	*true (1)*
true (1)	*false (0)*

3.4.6. For takrorlash operatori

For strukturasi sanovchi (counter) bilan bajariladigan takrorlashni bajaradi. Boshqa takrorlash bloklarida (*while, do/while*) takrorlash sonini nazorat qilish uchun ham sanovchini qo`llasa bo`lardi, bu holda takrorlanish sonini oldindan 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 soni kiritmaguncha takrorlashni bajarish kerak bo`lsa biz *while* li ifodalarni ishlatamiz. *For* da esa sanovchi ifodaning qiymati oshirilib (kamaytirilib) borilaveradi, 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.

//Ekranda o`zgaruvching qiymatini yozuvchi dastur, *For* ni ishlatadi.

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

Ekranda:

0

1

2

3

4

for strukturasi uch qismdan iboratdir. Ular nuqta vergul bilan bir-biridan ajratiladi. for ning ko`rinishi:

```
for(1- qism; 2- qism; 3- qism)
{
```

```
Takrorlanuvchi blok
}
```

1 - qism - e’lon va initsalizatsiya.

2 - qism - shartni tekshirish (o`zgaruvchini chegaraviy qiymat bilan solishtirish).

3 - qism - o`zgaruvchining qiymatini o`zgartirish.

Qismlarning bajarilish ketma-ketligi quyidagicha:

Dastlab 1 - qism bajariladi (faqat bir marta), keyin 2 - qismdagi shart tekshiriladi va agar u *true* bo`lsa takrorlanish bloki bajariladi va eng oxirda 3 - qismda 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 ga teng 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 boshlang`ich qiymatlar beriladi (initsializatsiya). 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 qiymatini 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 moslashuvchanligi 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.

Misol:

```
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 1 - qismda 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. Ya'ni bitta blokda joylashgan *for* struk-turalari 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 qilinib bo`lindi. Ikkinchi *for* da ishlatish mumkin emas. Bu masalani yechish uchun ikki hil 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

boshlang`ich qiymat beriladi xolos. *for* ning ko`rinishlaridan biri, bo`sh tanali *for* dir.

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

Buning yordamida biz dastur ishlashini sekinlashtirishimiz mumkin.

3.4.7. Boshqaruv operatorida *continue* va *break* operatorlaridan foydalanish

while, *dowhile*, *switch* va *for* strukturalarida *break* operatorini qo`llaganimizda dastur bajarilishi ushbu strukturalaridan chiqib ketadi va navbatdagi kelayotgan operatordan davom etadi. Bunda boshqaruv strukturalaridagi *break* dan keyin keluvchi ifodalar bajarilmaydi. Buni misolda ko`rib chiqaylik.

```
//break va for ni qo`llash
# include <iostream.h>
int main()
{
int h, k = 3;
for(h = 0; h < 10 ; h++)
{
cout << h << " ";
if (k == 6)
break;
cout << k++ << endl;
}
cout << "
for dan tashqarida: "
<< h
<< "
"
<< k
<< endl;
return (0);
```

```
}
```

Ekranda:

```
0   3
1   4
2   5
3
```

for dan tashqarida 3 6

if ning sharti bajarilgandan so`ng *break* dan keyin joylashgan cout << k++ << endl; operatori bajarilmadi. Biz o`zgaruvchilarni *for* dan tashqarida ham qo`llamoqchi bo`lganimiz uchun, ularni *for* dan oldin e'lon qildik. *continue* ifodasi while, do while yoki *for* ichida qo`llanilganda, takrorlanish tanasida *continue* dan keyin kelayotgan operatorlar bajarilmasdan, takrorlanishning yangi sikli (iteratsiyasi) boshlanadi. Bu jarayonni quyidagi dasturda ko`rib chiqaylik.

```
...
for (int e = 1 ; e<=10 ; ++e)
{
if ( (e%2) == 0 ) //juft son bo`lsa siklni o`tkazib yubor
continue;
cout << e << " ";
} ...
```

Ekranda:

1 3 5 7 9

Bu yerda *sontinue* va *break* ni ishlatish strukturali dasturlashga to`g`ri kelmaydi. Ular dasturni analiz qilishni murakkablashtirib yuboradi. Bular o`rniga strukturali dasturlash amallarini qo`llagan holda boshqaruv strukturalarining harakatini o`zgartirish mumkin. Lekin boshqa tarafdan albatta bu sakrash ifodalari ayni ishni bajaradigan strukturali dasturlash iboralaridan ko`ra ancha tezroq ishlaydi. Boshqaruv strukturalarini qo`llanilgan bir misol keltiraylik.

Dastur futbol o`yinlarining nechtasida durang, nechtasida birinchi va nechtasida ikkinchi komanda yutganini sanaydi.

```
// while - switch - cin.get - EOF ga misol
# include <iostream.h>
int main()
{
int natijaa = 0, // o`yin natijasi
durang = 0, // duranglar soni
birinchi = 0, // birinchi komanda yutug`i
ikkinchi = 0; // ikkinchi komanda yutug`i
cout << "Durang - d, birinchi komanda yutug`i - b,
ikkinchi komanda yutug`i - i\n"
<< "Tugatish uchun - EOF." << endl;
while ( ( natija = cin.get() ) != EOF )
{
switch (natija)
{
case 'D': // Katta harf uchun
case 'd': // Kichik harf uchun
durang++;
break; //
case 'B':
case 'b':
birinchi++;
break;
case 'I':
case 'i':
ikkinchi++;
break;
case '\n': //yangi satr
case '\t': //tabulatsiya
case ' ' : //va bo`shliqlarga e’tibor bermaslik
```

```
break;
default: // qolgan hamma harflarga javob:
cout << "Noto`g`ri harf kiritildi. Yangidan kiriting..."
break; // eng oxirida shart emas.
}
//end switch - switch bloki tugaganligi belgisi
}
//end while
cout << "\n\n\n Har bir hol uchun o`yinlar soni:"
<< "\n Durang: " << durang
<< "\n Birinchi komanda yutug`i: " << birinchi
<< "\n Ikkinchi komanda yutug`i: " << ikkinchi
<< endl;
return (0);
}
```

Bu dasturda uch xil holat uchun qo`llanuvchi harflarni kiritadi. *While* takrorlash strukturasining shart berilish qismida () qavslarga olingan ( natija = *cin.get()* ) qiymat berish amali birinchi bo`lib bajariladi. *cin.get()* funksiyasi klaviaturadan bitta harfni o`qib oladi va uning qiymatini int tipidagi natija o`zgaruvchisida saqlaydi. Harflar(character) odatda char tipidagi o`zgaruvchilarda saqlanadi. Lekin C++ da harflar istalgan integer(butun son) tip ichida saqlanishi mumkin, chunki kompyuter ichida harflar bir baytlik butun son tiplarida saqlanadi. Qolgan butun son tiplari esa bir baytdan kattadir. Shu sababli biz harflarni butun son (int) sifatida yoki harf sifatida ishlatishimiz mumkin.

```
cout << "L harfi int tipida " << static_cast<int>('L') << " ga teng." << enl;
```

Ekranda: L harfi int tipida 76 ga teng.

Demak, L harfi kompyuter xotirasida 76 qiymatiga ega. Hozirgi kunda kompyuterlarning asosiy qismi ASCII kodirovkada ishlaydi. (American Standard Code for Information Interchange - informatsiya ayrboshlash uchun amerika standart kodi) ASCII da 256 ta belgining raqami berilgan. Bu kodirovka 8 bit - bir

bayt joy oladi va o`z ichida asosan lotin alifbosi harflari berilgan. Milliy alifbolarni ifodalash uchun (arab, xitoy, yahudiy, kiril) yangi kodirovka – UNICODE ishlatilmoqda. Bunda bitta simvol yoki belgi ikkita bayt orqali beriladi. Ifodalanishi mumkin bo`lgan harflar soni 65536 tadir (2 ning 16 - darajasi). UNICODE ning asosiy noqulayligi - uning hajmidir. U asosan Internetga mo`ljallangan edi. Oldin ASCII bilan berilgan tekst hozir UNICODE da berilsa, uning hajmi ikki baravar oshib ketadi, ya’ni aloqa tarmoqlariga ikki marta ko`proq og`irlik tushadi.

Tenglashtirish ifodasining umumiy qiymati chap argumentga berilayotgan qiymat bilan tengdir. Buning qulaylik tarafi shundaki, biz d = f = g = 0; deb yozishimiz mumkin. Bunda oldin g nolga tenglashtiriladi keyin g = 0 ifodasining umumiy qiymati 0, f va d larga zanjir ko`rinishida uzatiladi.

Natija = cin.get() ifodasining umumiy qiymati EOF (End Of File – fayl oxiri) constantasi qiymati bilan solishtiriladi, va unga teng bo`lsa while takrorlash strukturasidan chiqiladi. EOF ning qiymati ko`pincha 1 bo`ladi. Lekin ASCII standarti EOF ni manfiy son sifatida belgilagan, ya’ni uning qiymati 1 dan farqli bo`lishi mumkin. Shu sababli 1 ga emas, EOF ga tenglikni test qilish dasturning universalligini, bir sistemadan boshqasiga osonlik bilan o`tishini ta’minlaydi. EOF ni kiritish uchun qo`llanuvchi maxsus tugmalar kombinatsiyasini bosadi. Bu bilan u "boshqa kiritishga ma’lumot yo`q" ekanligini bildiradi. EOF qiymati <iostream.h> da aniqlangan. DOS va DEC VAX VMS sistemalarida EOF ni kiritish uchun <ctrl-z> tugmalari bir vaqtda bosiladi. UNIX sistemalarida esa <ctrl-d> kiritiladi.

Qo`llanuvchi harfni kiritib, ENTER (RETURN) tugmasini bosgandan so`ng, cin.get() funksiyasi harfni o`qiydi. Bu qiymat EOF ga teng bo`lmasa, while tanasi bajariladi. Natijaning qiymati case etiketlarining qiymatlari bilan solishtiriladi. Masalan natija 'D' yoki 'd' ga teng bo`lganda durang o`zgaruvchisining qiymati bittaga oshiriladi. Keyin esa break orqali switch tanasidan chiqiladi. *switch* ning bir xususiyati shundaki, ifodalar bloki {} qavslarga olinishi shart emas. Blokning kirish nuqtasi case, chiqish nuqtasi esa break operatoridir.

```
case '\n':
case '\t':
case ' ' :
break;
```

Yuqoridagi dastur bloki qo`llanuvchi yanglish kiritgan yangi satr,tabulatsiya va bo`shliq belgilarini filtrlash uchun yozilgan. Eng oxirgi break ning majburiy emasligining sababi shuki, break dan so`ng boshqa operatorlar yo`q. Demak, break qo`yilmagan taqdirda ham hech narsa bajarilmaydi. EOF kiritilgandan so`ng while tugaydi, o`zgaruvchilar ekranga bosib chiqariladi.

Masala: Talabaning n ta olgan baholariga qarab uning o`qish sifatini aniqlovchi dasturini tuzing. Buning uchun dasturda o`quvchining olgan minimal bahosi aniqlanadi

```
# include <iostream.h>
void main()
{
int I,n,min,p;
while (1)
{Cout<<“Baholar soni=”; Cin>>n;};
If (n>0) break;
Cout<<(“Xato! n>0 bo`lishi kerak! \n”);
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) cout break;
switch(min)
case 0:cout<<”Baho noto`g`ri kiritilgan”;break;
case 2:cout<<”Talaba  2 baholi”;break;
case 3:cout<<” Talaba 3 baholi”;break;
case 4:cout<<” Talaba  4 baholi”;break;
case 5:cout<<” Talaba  5 baholi”;break;
}
```

Tayanch so’z va iboralar

Takrorlanuvchi algoritm, sikl, break, while, do…while, o’tish operatori, sanash, o’zgaruvchi, argument, Inkor operatori.

Mavzuga oid savollar va topshiriqlar

1. Takrorlanuvchi algoritmga ta’rif bering.
2. Takrorlanuvchi algoritmlarning qanaqa turlari bo’lishi mumkin?
3. *for* operatori qachon qo’llaniladi?
4. *Do… while* operatori qachon qo’llaniladi?
5. *while* operatori qachon qo’llaniladi?
6. *Do… while* operatori bilan *while* operatorining farqi?