17-MA’RUZA. FUNKSIYANI QAYTA YUKLASH. KELISHUV BO’YICHA ARGUMENTLAR

1. Nomlar fazosi. Ko‘rinishi sohasi.

Ko‘rinishi sohasi (scope) dasturning obyektini ishlatish mumkin bo‘lgan qismini ifodalaydi. Odatda, ko‘rinish sohasi sistemali qavs ichida joylashgan kod bloki bilan cheklanadi. Ko‘rinish sohasiga qarab, yaratilgan obyektlar **global**, **lokal** yoki **avtomatik** bo‘lishi mumkin.

Dastlab, ikkita atamani tushunishimiz kerak: **ko‘rinish sohasi** va **yashash vaqti**. Ko‘rinish sohasi o‘zgaruvchini qayerda ishlatishni aniqlaydi. Yashash vaqti (yoki "yashash muddati") o‘zgaruvchining qayerda yaratilganligini va u yo‘q qilinishini aniqlaydi. Ushbu ikkita tushuncha bir-biriga bog‘liqdir.

Blok ichida aniqlangan o‘zgaruvchilar **lokal** o‘zgaruvchilar deyiladi. Lokal o‘zgaruvchilar avtomatik ishlash muddatiga ega: ular aniqlanish nuqtasida yaratiladi (va agar kerak bo‘lsa, ishga tushiriladi) va blokdan chiqqandan keyin yo‘q qilinadi. Lokal o‘zgaruvchilar lokal ko‘rinish sohasiga (yoki "blok") ega, ya’ni ular e’lon qilingan nuqtadan kelib chiqadi va ular aniqlangan blokning oxirida chiqadi.

Masalan, quyidagi dasturni ko‘rib chiqaylik:

```
#include <iostream>
using namespace std;
int main()
{
   int x=4;
   double y=5.0;
   cout<<x;
   return 0;
}
```

Asosiy funksiya bloki ichida x va y o‘zgaruvchilar aniqlanganligi sababli, ikkalasi ham main () bajarilishini tugatgandan so‘ng yo‘q qilinadi.

Ichki bloklar ichida belgilangan o‘zgaruvchilar ichki blok tugashi bilanoq yo‘q qilinadi:

```
#include <iostream>
int main() // tashqi blok
{
   int m=4; // bu yerda m o‘zgaruvchisi e'lon qilinyapti va initsializatsiya qilinyapti
```

```
    { // ichki blokning boshlanishi
        double k=5.0; //bu yerda k o‘zgaruvchisi yaratiladi va ishga tushiriladi
    } // k ko‘rinish sohasidan chiqib ketadi va bu yerda yo‘q qilinadi

    // Bu yerda k o‘zgaruvchisini ishlatib bo‘lmaydi.
    k  = k+1; //Bu xato
        return 0;
}
```

Bunday o‘zgaruvchilardan faqat ular aniqlangan bloklar ichida foydalanish mumkin. Har bir funksiya o‘z blokiga ega bo‘lganligi sababli, bitta funksiyadagi o‘zgaruvchilar boshqa funksiyadagi o‘zgaruvchilarga ta’sir qilmaydi:

```
#include <iostream>
using namespace std;
void SFunk()
{
    int value=5;
    // value o‘zgaruvchisidan faqat shu blokda foydalanish mumkin
} // value o‘zgaruvchisi ko‘rinishi sohasidan chiqdi va u yo‘q qilinadi

int main()
{
    // value o‘zgaruvchisidan bu blokda foydalanib bo‘lmaydi
    SFunk();
    return 0;
}
```

Turli xil funksiyalar bir xil nomdagi o‘zgaruvchilar yoki parametrlarni o‘z ichiga olishi mumkin. Bu yaxshi, chunki ikkita mustaqil funksiya o‘rtasidagi ziddiyatlarni nomlash ehtimoli haqida tashvishlanishingiz shart emas. Quyidagi misolda ikkala funksiya ham x va y o‘zgaruvchilarga ega. Ular hatto bir-birlarining mavjudligini bilishmaydi:

```
#include <iostream>
using namespace std;
// add() funksiyasi
int add(int x, int y)
```

```
{
    return x + y;
}

// main() ichida ham x va y o‘zgaruvchilari ishlatilgan
int main()
{
    int x = 5; // x o‘zgaruvchisi e'lon qilingan
    int y = 6;
    cout << add(x, y) << endl; //main() funksiyasining x qiymati add() funksiyasining x o‘zgaruvchisiga ko‘chiriladi
    return 0;
}
```

Ichki bloklar ular belgilanadigan tashqi blokning bir qismi hisoblanadi. Shunday qilib, tashqi blokda aniqlangan o‘zgaruvchilar ichki blok ichida ham ko‘rish mumkin:

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

    int x=5;

    { // ichki blokning boshlanishi
        int y=7;
        // x va y dan foydalanish mumkin
        cout << x << " + " << y << " = " << x + y;
    } // y o‘zgaruvchisi yo‘q qilinadi

    // y o‘zgaruvchisini bu yerda ishlatish mumkin emas, chunki u allaqachon yo‘q qilingan!

    return 0;
}
```

Nomlarni yashirish. Ichki blok ichidagi o‘zgaruvchi tashqi blok ichidagi o‘zgaruvchiga o‘xshash nomga ega bo‘lishi mumkin. Bu sodir bo‘lganda, ichki

(ichki) blokdagi o‘zgaruvchi tashqi o‘zgaruvchini "yashiradi". Bunga nomlarni yashirish deyiladi:

```
#include <iostream>
using namespace std;
int main()
{ // tashqi blok
  int a=5;

  if (a >= 5)
  { // ichki blok
        int a; // a o‘zgaruvchisi ichki blokda yashirilmoqda

        // a identifikatori endi ichki o‘rnatilgan a o‘zgaruvchiga ishora qiladi.
      // a tashqi o‘zgaruvchisi vaqtincha yashiringan

        a = 10; //Bu yerda tashqi emas, balki ichki a o‘zgaruvchisiga qiymat berilgan

        cout << a << endl;
  } // ichki a o‘zgaruvchisi o‘chirildi

  // a identifikatori yana tashqi o‘zgaruvchiga ishora qiladi

  cout << a << endl;
  return 0;
}
```

Imkon qadar nomlarni yashirishga umuman yo‘l qo‘ymaslik kerak, chunki bu juda chalkash holatlarda olib kelishi mumkin. Iloji boricha tashqi o‘zgaruvchilar bilan bir xil nomdagi ichki o‘zgaruvchilarni ishlatishdan saqlanish lozim.

2. Lokal va global o‘zgaruvchilar

Global o‘zgaruvchilar. Global o‘zgaruvchilar dastur faylida har qanday funksiyalardan tashqarida aniqlanadi va har qanday funksiya tomonidan ishlatilishi mumkin.

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

```
void print();
int n = 5; //Global o‘zgaruvchi
int main()
{
   print();                    // n=6
   n++;
   cout << "n=" << n << endl;   // n=7
   return 0;
}
void print()
{
   n++;
   cout << "n=" << n << endl;
}
```

Bu yerda n o‘zgaruvchisi global va har qanday funksiyadan foydalanish mumkin. Bundan tashqari, har qanday funksiya o‘z qiymatini o‘zgartirishi mumkin.

3. Funksiyani qayta yuklash

Ayrim algoritmlar berilganlarning har xil turdagi qiymatlari uchun qo‘llanishi mumkin. Masalan, ikkita sonning maksimumini topish algoritmida bu sonlar butun yoki haqiqiy turda bo‘lishi mumkin. Bunday hollarda bu algoritmlar amalga oshirilgan funksiyalar nomlari bir xil bo‘lgani ma’qul. Bir nechta funksiyani bir xil nomlash, lekin har xil turdagi parametrlar bilan ishlatish *funksiyani qayta yuklash* deyiladi.

Kompilyator parametrlar turiga va soniga qarab mos funksiyani chaqiradi. Bunday amalni “*hal qilish amali*” deyiladi va uning maqsadi parametrlarga ko‘ra aynan (nisbatan) to‘g‘ri keladigan funksiyani chaqirishdir. Agar bunday funksiya topilmasa kompilyator xatolik haqida xabar beradi. Funksiyani aniqlashda funksiya qaytaruvchi qiymat turining ahamiyati yo‘q. Misol:

```
#include <iostream>
using namespace std;
int Max(int,int);
char Max(char,char);
float Max(float,float);
int Max(int,int,int);
int main()
{
   int a,b;
```

```
    char c,d;
    int k;
    float x,y;
    cin>>a>>b>>k>>c>>d>>x>>y;
    cout<<Max(a,b)<<endl;
    cout<<Max(c,d)<<endl;
    cout<<Max(a,b,k)<<endl;
    cout<<Max(x,y);
    return 0;
}

int Max(int i,int j)
{
    return (i>j)?i:j;
}
char Max(char s1,char s2)
{
    return (s1>s2)?s1:s2;
}
float Max(float x,float y)
{
    return (x>y)?x:y;
}
int Max(int i,int j,int k)
{
    if(i>j)
        if(i>k)
        return i;
        else
            return k;
        if(j>k)
            return j;
        else
            return k;
}
```

Agar funksiya chaqirilishida argument turi uning prototipidagi xuddi shu o‘rindagi parametr turiga mos kelmasa, kompilyator uni parametr turiga keltirilishga harakat qiladi - bool va char turlarini int turiga, float turini double turiga va int turini double turiga o‘tkazishga.

Qayta yuklanuvchi funksiyalardan foydalanishda quyidagi qoidalarga rioya qilish kerak:

1) qayta yuklanuvchi funksiyalar bitta koʻrinish sohasida boʻlishi kerak;

2) qayta yuklanuvchi funksiyalarda kelishuv boʻyicha parametrlar ishlatilsa, bunday parametrlar barcha qayta yuklanuvchi funksiyalarda ham ishlatilishi va ular bir xil qiymatga ega boʻlish kerak;

3) agar funksiyalar parametrlarining turi faqat «const» va '&' belgilari bilan farq qiladigan boʻlsa, bu funksiyalar qayta yuklanmaydi.

4. Kelishuv bo‘yicha argumentlar

C++ tilida funksiya chaqirilganda ayrim argumentlarni tushirib qoldirish mumkin. Bunga funksiya prototipida ushbu parametrlarni kelishuv bo‘yicha qiymatini ko‘rsatish orqali erishish mumkin. Masalan, quyida prototipi keltirilgan funksiya turli chaqirishga ega bo‘lishi mumkin:

```
//funksiya prototipi
void Butun_Son(int I,bool Bayroq=true,char Blg='\n');
 //funksiyani chaqirish variantlari
Butun_Son(1,false,'a');
Butun_Son(2,false);
Butun_Son(3);
```

Birinchi chaqiruvda barcha parametrlar mos argumentlar orqali qiymatlarini qabul qiladi, ikkinchi holda I parametri 2 qiymatini, bayroq parametri false qiymatini va Blg o‘zgaruvchisi kelishuv bo‘yicha ‘\n’ qiymatini qabul qiladi.

Kelishuv bo‘yicha qiymat berishning bitta sharti bor - parametrlar ro‘yxatida kelishuv bo‘yicha qiymat berilgan parametrlardan keyingi parametrlar ham kelishuv bo‘yicha qiymatga ega bo‘lishlari shart. Yuqoridagi misolda I parametri kelishuv bo‘yicha qiymat qabul qilingan holda, Bayroq yoki Blg parametrlari qiymatsiz bo‘lishi mumkin emas. Misol tariqasida berilgan sonni ko‘rsatilgan aniqlikda chop etuvchi programmani ko‘raylik. Qo‘yilgan masalani yechishda sonni darajaga oshirish funksiyasi - pow() va suzuvchi nuqtali uzun sondan modul olish fabsl() funksiyasidan foydal-niladi. Bu funksiyalar prototipi «math.h» sarlavha faylida joylashgan.

```
#include <iostream>
#include <math.h>
using namespace std;
void Chop_qilish(double Numb, double Aniqlik=1, bool Bayroq = true)
{
   if(!Bayroq)
      Numb=fabsl(Numb);
   Numb=(int)(Numb*pow(10,Aniqlik));
   Numb=Numb/pow(10,Aniqlik);
   cout<<Numb<<'\n';
}

int main()
{
```

```
    double Mpi=-3.141592654;
    Chop_qilish(Mpi,4,false);
    Chop_qilish(Mpi,2);
    Chop_qilish(Mpi);
    return 0;
}
```

Programmada sonni turli aniqlikda (Aniqlik parametri qiymati orqali) chop etish uchun har xil variantlarda Chop_qilish() funksiyasi chaqirilgan. Programma ishlashi natijasida ekranda quyidagi sonlar chop etiladi:

-3.1415
-3.14
-3.1

Parametrning kelishuv bo‘yicha beriladigan qiymati o‘zgarmas, global o‘zgaruvchi yoki qandaydir funksiya tomonidan qaytaradigan qiymat bo‘lishi mumkin.

5. Rekursiv funksiyalar

Rekursiya - bu nafaqat ilm-fan sohasida, balki kundalik hayotda ham uchraydigan juda keng tarqalgan hodisa.

Dasturlashda rekursiya funksiyalar bilan chambarchas bog‘liq, aniqrog‘i dasturlashdagi funksiyalar tufayli rekursiya yoki rekursiv funksiya kabi tushunchalar mavjud. Oddiy so‘zlar bilan aytganda, rekursiya - bu funksiya qismini o‘zi orqali belgilash, ya’ni o‘zini to‘g‘ridan-to‘g‘ri (tanasida) yoki bilvosita (boshqa funksiya orqali) chaqiradigan funksiya. Odatda rekursiv muammolarga sonning faktorialini topish, Fibonachchi raqamini topish va hokazolarni keltirish mumkin. Bu kabi masalalarni sikllar yordamida ham hal qilish mumkin. Umuman aytganda, iterativ ravishda yechilgan hamma narsani rekursiv, ya’ni rekursiv funksiya yordamida hal qilish mumkin.

C++ da **rekursiv** funksiya (yoki shunchaki "rekursiya") o‘zini o‘zi chaqiradigan funksiya.

Masalan:

```
#include <iostream>
using namespace std;
void countOut(int count1)
{
    cout << "push " << count1 << '\n';
    countOut(count1-1); //countOut () funksiyasi o‘zini rekursiv chaqiradi
```

```
}
int main()
{
   countOut(4);
   return 0;
}
```

countOut(4) ga murojaat qilish “push 4” yozuvini bosib chiqaradi va keyin countOut (3) ni chaqiradi. countOut (3) “push 3” ni bosib chiqaradi va countOut (2) ga murojaat qiladi.

Rekursiyani tugatish sharti. Rekursiv funksiya chaqiruvlari odatdagi funksiya murojaatlari singari ishlaydi. Shu bilan birga, yuqoridagi dastur oddiy funksiyalar va rekursivlar o‘rtasidagi eng muhim farqni aks ettiradi: siz rekursiyani tugatish shartini belgilashingiz kerak, aks holda funksiya cheksiz marta bajariladi.

Rekursiyani tugatish sharti - bu bajarilgandan so‘ng rekursiv funksiyani o‘zi chaqirishni to‘xtatadigan shart. Ushbu holat odatda if ifodasini ishlatadi.

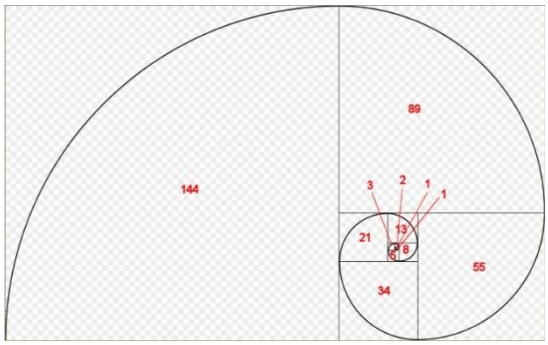
Bu yerda yuqoridagi funksiyaga misol keltirilgan, ammo bu yerda rekursiya tugashi sharti ham mavjud:

```
#include <iostream>
using namespace std;
void countOut(int count1)
{
   cout << count1<< "-chiqish " << '\n';
   if (count1 > 1) // chiqish sharti
      countOut(count1-1);
   cout <<  count1<< "-kirish " << '\n';
}

int main()
{
   countOut(4);
   return 0;
}
```

Rekursiv algoritmlar. Eng mashhur matematik rekursiv algoritmlardan biri bu Fibonachchi ketma-ketligi. Fibonachchi ketma-ketligini tabiatda ham ko‘rish mumkin: daraxt shoxlari, spiral kabuklar, ananas mevalari, ochiladigan fern va boshqalar.

12-rasm. Fibonachchi sonlari



Fibonachchi raqamlarining har biri berilgan son joylashgan kvadratning gorizontal tomonining uzunligi. Fibonachchi raqamlari matematik jihatdan quyidagicha aniqlanadi:

agar n = 0 bo‘lsa, F(1) = 0

agar n = 1, F(2)=1,

agar n > 1, F(n-1) + F(n-2),

Shunday qilib, Fibonachchi sonini hisoblash uchun rekursiv funksiyani yozish juda oson:

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

int fibonacci(int number)
{
   if (number == 0)
      return 0;
   if (number == 1)
      return 1; //
   return fibonacci(number-1) + fibonacci(number-2);
}
// Dastlabki 13 ta Fibonachchi sonini topish
int main()
{
   for (int count=0; count < 13; ++count)
      cout << fibonacci(count) << " ";
   return 0;
```

}

Rekursiya va iteratsiya. Rekursiv funksiyalar haqida eng koʻp soʻraladigan savol: "Nima uchun rekursiv funksiyadan foydalanish kerak, chunki bu masalalarni for sikli yoki while sikli yordamida ham bajarish mumkin?" Ma'lum boʻlishicha, har doim rekursiv masalani takroriy ravishda hal qilishingiz mumkin. Biroq, ahamiyatsiz boʻlmagan holatlar uchun rekursiv versiyani yozish va oʻqish koʻpincha ancha osonlashadi. Masalan, n-Fibonachchi raqamini hisoblash funksiyasini takrorlash operatorlari yordamida yozish mumkin, ammo bu murakkab boʻladi.

Iteratsion funksiyalar (**for** yoki **while** sikllaridan foydalanadigan funksiyalar) deyarli har doim oʻzlarining rekursiv oʻxshashlariga qaraganda samaraliroq. Buning sababi shundaki, har safar funksiya chaqirilganda ma'lum miqdordagi resurslar sarflanadi. Iterativ funksiyalar ancha kam resurs sarf qiladi.

Bu takrorlanadigan funksiyalar har doim eng yaxshi variant degani emas. Umuman olganda, rekursiya, agar quyidagilarning aksariyati toʻgʻri boʻlsa, yaxshi tanlovdir:

- rekursiv kodni amalga oshirish ancha oson;
- rekursiya chuqurligini cheklash mumkin;
- algoritmning takrorlanadigan versiyasi ma'lumotlar toʻplamini boshqarishni talab qiladi;
- bu dasturning ishlashiga bevosita ta'sir koʻrsatadigan muhim kod emas.

6. Funksiyalardan foydalanish sabablari

Dastlabki boshlovchi dasturchilarda koʻpincha "Masalani funksiyalarsiz bajarish va barcha kodlarni toʻgʻridan-toʻgʻri main() funksiyasiga qoʻyish mumkinmi?" degan savol paydo boʻladi. Agar dastur kodingiz atigi 10-20 satrdan iborat boʻlsa, unda siz buni qila olasiz. Jiddiy eslatma sifatida, funksiyalar kodni murakkablashtirishga emas, balki soddalashtirishga qaratilgan. Ularning notrivial dasturlarda juda foydali boʻlgan bir qator afzalliklar bor.

Struktura. Dasturlar hajmi/murakkabligi oshib borgan sari, barcha kodlarni main() ichida saqlash qiyin boʻladi. Funksiya mini-dasturga oʻxshaydi, biz uni kodning qolgan qismi bilan asosiy dasturdan alohida yozishimiz mumkin. Bu sizga murakkab vazifalarni kichikroq va sodda vazifalarga ajratishga imkon beradi, bu esa dasturning umumiy murakkabligini keskin pasaytiradi.

Takror foydalanish. Funksiya e'lon qilingandan soʻng, uni koʻp marta chaqirish mumkin. Bu kodning takrorlanishiga yoʻl qoʻymaydi va kodni nusxalash/joylashtirishda xatolar ehtimolini kamaytiradi. Funksiyalar boshqa dasturlarda ham ishlatilishi mumkin, bu har safar noldan yozilishi kerak boʻlgan kod miqdorini kamaytiradi.

Testlash. Funksiyalar keraksiz kodni olib tashlaganligi sababli, uni testlash osonroq boʻladi va funksiya mustaqil birlik boʻlgani uchun, uning ishlashiga ishonch hosil qilish uchun uni bir marta sinab koʻrishimiz kerak, keyin uni sinovdan oʻtkazmasdan (bu funksiyaga oʻzgartirish kiritmagunimizcha) qayta-qayta ishlata olamiz.

Modernizatsiya. Agar dasturga oʻzgartirish kiritishingiz yoki uning funksiyasini kengaytirishingiz kerak boʻlsa, unda funksiyalar juda yaxshi imkoniyatdir. Ularning yordami bilan hamma joyda ishlashlari uchun ularni bitta joyda oʻzgartirishingiz mumkin.

Abstraksiya. Funksiyadan foydalanish uchun biz uning nomini, kirish ma'lumotlarini, chiqish ma'lumotlarini va funksiya qayerda joylashganligini bilishimiz kerak. Uning qanday ishlashini bilishimiz shart emas. Bu boshqalar tushunishi mumkin boʻlgan kod yozish uchun juda foydalidir (masalan, C++ standart kutubxonasi va undagi barcha narsalar shu prinsip asosida tuzilgan).

Har safar kiritish yoki chiqarish operatori uchun std::cin yoki std::cout deb nomlaganimizda, biz C++ standart kutubxonasidan yuqoridagi barcha tushunchalarga amal qiladigan funksiyadan foydalanamiz.

Funksiyalardan samarali foydalanish. Yangi boshlovchilar duch keladigan eng keng tarqalgan muammolardan biri bu funksiyalarni qayerda, qachon va qanday qilib samarali ishlatishni tushunishdir. Funksiyalarni yozish uchun ba'zi bir asosiy koʻrsatmalar:

1-tavsiya. Dasturda bir necha marta paydo boʻlgan kod funksiya sifatida yaxshi yozilgan. Masalan, agar biz xuddi shu tarzda foydalanuvchidan bir necha marta ma'lumotlarni olsak, bu alohida funksiyani yozish uchun ajoyib imkoniyat.

2-tavsiya. Biron bir narsani saralash uchun ishlatiladigan kod alohida funksiya sifatida yozilgan yaxshiroqdir. Masalan, bizda saralash kerak boʻlgan narsalar roʻyxati boʻlsa, biz saralash funksiyasini yozamiz, bu yerda biz saralanmagan roʻyxatni oʻtkazamiz va saralangan joyni olamiz.

3-tavsiya. Funksiya bitta (va faqat bitta) vazifani bajarishi kerak.

4-tavsiya. Agar funksiya juda katta, murakkab yoki tushunarsiz boʻlib qolsa, uni bir nechta kichik funksiyalarga boʻlish kerak. Bunga kodni qayta ishlash deyiladi.