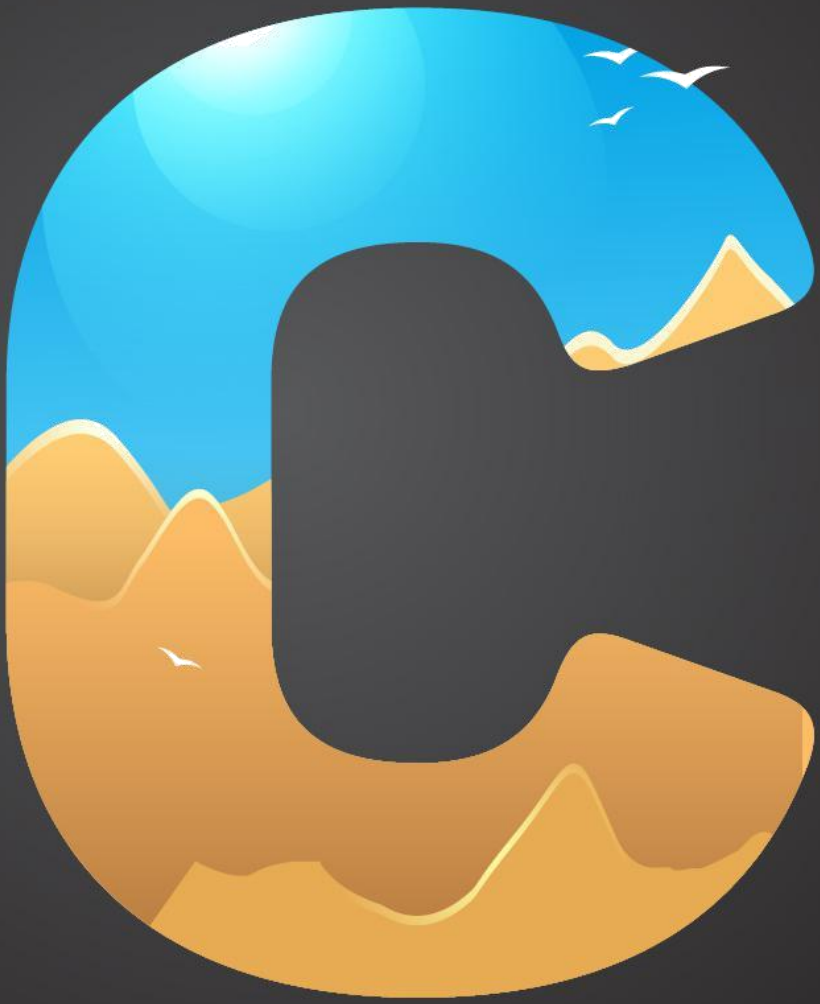


ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Dərs №11

C
dilində proqramlaşdırma

Mündəricat

1. Rekursiya ilə tanışlıq

Rekursiya – bu bir proqramlaşdırma üsuludur ki, burada proqram özunu birbaşa və ya dolayısilə çağırır.

Bir qayda olaraq, təcrübəsiz proqramçı rekursiya haqqında bildikdə bir az heyrətlənəcək. İlk fikir – bu mənasızdır!!! Bu cür ardıcıl çağırışlar, ilanın özünü udamağa çalışması kimi sonsuz dövrə çevriləcək və ya proqram icra zamanı kompüter yaddaşının bütün resurslarını "udacağı" üçün xəta baş verəcək.

Lakin rekursiya – bacrıqla və düzgün istifadə edildiyi zaman proqramçıya bir çox mürəkkəb məsələləri həl etməkdə kömək edəcək.

Rekursiyaya nümunə:

Adətən ənənəvi olaraq rekursiyaya faktorialın hesablanmasını nümunə kimi göstərirlər.

Biz də bu ənənəni pozmayacağıq.

Əvvəlcə faktorialın nə olduğunu xatırlayaq. Faktorial «!» işərəsi ilə göstərilir və növbəti şəkildə hesablanır:

```
N! = 1 * 2 * 3 * …  * N
```

Başqa sözlə, faktorial 1-dən N-ə qədər natural ədədlərin hasilidr. Yuxarıda verilən düsturdan növbəti qanunauyğunluğu əldə etmək olar:

```
N! = N * (N-1)!
```

Buradan göründüyü kimi biz faktorialı elə faktorialın özündən hesablaya bilərik. Elə burda biz tələyə düşürük. Bizim tapıntımız, ilk baxışda, tamamilə faydasız görünür, belə ki, məlum olmayan anlayış elə onun kimi məlum olmayan anlayışla təyin edilir və sonsuz dövr alınır. Bu vəziyyətdən çıxış yolu, əgər faktorialın təyininə növbəti faktı əlavə edəriksə, tez tapılacaq:

```
1!=1
```

Artıq biz istənilən ədədin faktorialını hesablaya bilərik. N! = N * (N–1)! düsturunu bir neçə dəfə və 1! = 1 düsturunu bir dəfə tətbiq etməklə 5!-lı hesablamağa çalışaq.

```
5! = 5 * 4! = 5 * 4 * 3! = 5 * 4 * 3 * 2! = 5 * 4 * 3 * 2 * 1! = 5 * 4 * 3 * 2 * 1
```

Bu alqoritmi C dilində necə yaza bilərik? Gəlin rekursiv funksiyanı reallaşdırmağa çalışaq:

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

long int Fact(long int N)
{
   //əgər 0!-ı hesablamağa cəhd edilərsə,
   if (N < 1) return 0;
   //1! hesablanarsa,
   //elə burada rekursiyadan çıxılır
       else if (N == 1) return 1;
   //digər istənilən ədəd funksiyanı N-1 düsturu ilə
   yenidən çağırır
       else return N * Fact(N-1);
}
```

```
void main()
{
   long number=5;
   //Rekursiv funksiyanın ilk çağırılması
   long result=Fact(number);
   cout<<"Result "<<number<<"! is - "<<result<<"\n";
}
```

Göründüyü kimi, hər şey elə də çətin deyildir. Misalı daha yaxşı başa düşmək üçün proqramın mətnini Visual Studio-ya köçürüb sazlayıcı ilə addım addım gözdən keçirə bilərsiniz.

2. Rekursiya və ya iterasiya?

Dərsin əvvəlki bölməsini öyrəndikdən sonra yəqin ki, rekursiyanın nəyə lazım olduğu sualı ortay çıxır. Belə ki, faktorialın hesablanmasını iterasiya ilə də reallaşdırmaq olar və bu heç də çətin deyildir:

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

long int Fact2(long int N)
{
       long int F = 1;
       //dövr faktorialın hesablanmasını həyata keçirir
       for (long int i=2; i<=N; i++)
                     F *= i;
       return F;
}

void main()
{
   long number=5;
   long result=Fact2(number);
   cout<<"Result "<<number<<"! is - "<<result<<"\n";
}
```

Bu alqoritm proqramçı üçün daha real görünür. Həqiqətdə isə bu heç də belə deyildir. Nəzəri baxımdan rekursiv reallaşdırıla bilən hər bir alqoritm çox rahatlıqla iterativ reallaşdırıla bilər. Biz artıq buna əmin olduq.

Lakin bu heç də belə deyildir. Rekursiya hesablamanı iterasiyaya nisbətən daha keç icra edir. Bundan başqa rekursiya icra prosesində nisbətən daha çox yaddaş tələb edir.

Bu rekursiyanın lazımsız olduğunu göstərirmi? Heç bir halda!!! Bir çox məsələlər vardır ki, onların rekursiv həlli daha dəqiq və gözəldir, iterativ isə - çətin, böyük və qeyri-təbiidir.

Bu halda sizin vəzifəniz nəinki rekursiv və ya iterativ əməliyyatlar aparmağı bacarmaq deyil, həmçinin müstəsna hallarda hansı yanaşmanı tətbiq etməyə intuitiv qərar verməkdən ibarətdir.

Deyə bilərik ki, rekursiyanın ən yaxşı tətbiqi – bu növbəti xüsusiyyətə malik məsələnin həllidir: məsələnin həlli elə bu cür məsələlərin həllinə gətirilir ki, ölçüsü az olsun və asan həll edilmiş olsun.

Bu yolda sizə müvəffəqiyyətlər arzulayırıq! Deyildiyi kimi: «Rekursiyanı anlamaq üçün, sadəcə rekursiyanı anlamaq lazımdır».

3. Sürətli çeşidləmə

«Sürətli çeşidləmə» - 40 il əvvəl işlənmişdir və daha çox tətbiq edilir. Prinsipcə ən səmərəli çeşidləmə alqoritmidir. Bu metod masssivin hissələrə ayrılmasına əsaslanır. Ümumi sxemi növbəti şəkildədir:

1. Massivdə hər hansı bir a[i] istinad elementi götürülür.
2. Massivin bölünməsi funksiyası işə salınır, bu funksiya a[i]-dən kiçik və ya bərabər olan bütün açarları ondan sola, böyük və ya bərabər olanları isə sağa yerləşdirir, beləliklə massiv iki hissədən ibarət olur, soldakı elementlər sağdakılardan kiçik olur.
3. Əgər altmasivdə ikidən artıq element olarsa, bu funksiyanı onlar üçün də rekursiv işə salırıq.
4. Sonda tamamilə çeşidlənmiş ardıcıllıq alınır.

Alqoritmi daha ətraflı gözdən keçirək.

Massivi iki hissəyə ayıraq.

Giriş verilənləri: a[0]...a[N] massivi və bölünmənin aparıldığı p elementi.

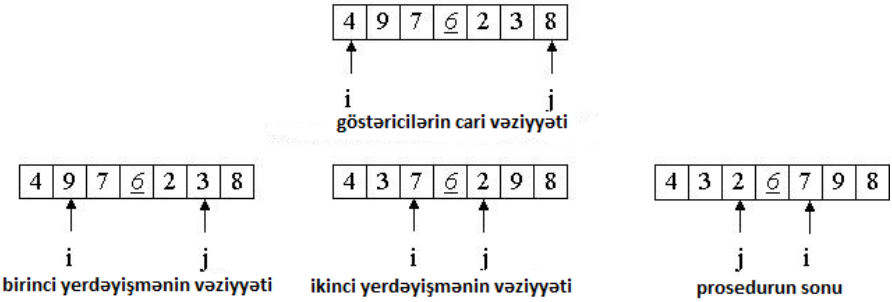
1. İki göstərici təyin edək: i və j. Alqoritmin əvvəlində onlar uyğun olaraq ardıcıllığın ən sol və ən sağ indeksini ifadə edirlər.
2. i göstəricisini a[i] >= p elementi tapılana qədər addım-addım massivin sonuna doğru hərəkət etdiririk.

3. Sonra isə analoji olaraq j göstəricisini a[j] <= p elementi tapılana qədər massivin sonundan əvvəlinə doğru addım-addım hərəkət etdiririk.

4. Daha sonra əgər i <= j olarsa, a[i] və a[j] elementlərinin yerlərini dəyişdiririk və i və j-ni eyni qayda ilə hərəkət etdiririk.

5. 3 addımını i <= j olana qədər təkrarlayırıq.

İstinad elementinin p = a[3] olduğu şəkilə baxaq.



Massiv iki hissəyyə ayrılmışdır: bütün soldakı elementlər p-dən kiçik və ya bərabər, sağdakılar isə böyük və ya bərabərdirlər.

Proqram nümunəsi.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

template <class T>

void quickSortR(T a[], long N) {
  // Girişdə a[] massivi, a[N] - onun sonuncu elementidir.
  // Göstəriciləri cari yerlərdə yerləşdirmək
  long i = 0, j = N;
  T temp, p;

  p = a[ N/2 ];      // mərkəzi element
  // Bölünmə proseduru
  do {
```

```
    while ( a[i] < p ) i++;
    while ( a[j] > p ) j--;

    if (i <= j){
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
        i++;
        j--;
    }

  }while ( i<=j );


  // Çeşidləmək üçün nə isə varsa, rekursiv çağırış
  if ( j > 0 ) quickSortR(a, j);
  if ( N > i ) quickSortR(a+i, N-i);
}

void main(){
       srand(time(NULL));
       const long SIZE=10;
       int ar[SIZE];

       // çeşidlənmədən əvvəl
       for(int i=0;i<SIZE;i++){
              ar[i]=rand()%100;
              cout<<ar[i]<<"\t";
       }
       cout<<"\n\n";

       quickSortR(ar,SIZE-1);

       // çeşidləndikdən sonra
       for(int i=0;i<SIZE;i++){
              cout<<ar[i]<<"\t";
       }
       cout<<"\n\n";
}
```

Rekursiv alqoritm.

1. Massivin mərkəzində p istinad elementi götürmək
2. Massivi bu elementə nisbətən iki hissəyə ayırmaq
3. Əgər p-dən soldakı altmassiv birdən çox element ehtiva edirsə, onun üçün quickSortR funksiyasını çağırmaq.
4. Əgər p-dən sağdakı altmassiv birdən çox element ehtiva edirsə, onun üçün quickSortR funksiyasını çağırmaq.

4. İkilik axtarış

Əvvəlki dərsdə biz xətti axtarış alqoritminə baxmışdıq, lakin bu massivdə axtarış aparmağın yeganə üsulu deyil. Əgər bizim sıralanmış massivimiz varsa, bu halda ikilik axtarış daha səmərəlidir.

İkilik axtarış nəzəriyyəsi.

Hesab edək ki, Lb və Ub dəyişənləri massivin uyğun olaraq sol və sağ sərhədlərini təyin edirlər. Axtarışı biz həmişə massiv parçasının mərkəzi elementini analiz edərək başlayacağıq. Əgər axtardığımız qiymət mərkəzi elementdən kiçik olarsa, onda biz axtarışı massivin baxdığımız elementindən kiçik olan yuxarı yarım parçasında davam etdiririk. Başqa sözlə, Ub-nin qiyməti (M (mərkəzi element) - 1) olur və növbəti iterasiyada massivin yarsı ilə işləyirik. Beləliklə də, hər yoxlama nəticəsində massivin axtarış sahəsini iki dəfə azaldırıq. Bizim misalda, birinci iterasiyadan sonra axtarış sahəsi yalnız 3 element ehtiva edir, ikincidən sonra isə yalnız bir element. Beləliklə, əgər massivin uzunluğu 6 olarsa, lazım olan elementi tapmaq üçün bizə 3 iterasiya kifayətdir.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

int BinarySearch (int A[], int Lb, int Ub, int Key)
{
	int M;
	while(1){
		M = (Lb + Ub)/2;
		if (Key < A[M])
			Ub = M - 1;
		else if (Key > A[M])
			Lb = M + 1;
		else
			return M;

		if (Lb > Ub)
			return -1;
	}
}

void main(){
	srand(time(NULL));
	const long SIZE=10;
	int ar[SIZE];
	int key,ind;

	// Çeşidlənmədən əvvəl
	for(int i=0;i<SIZE;i++)
	{
		ar[i]=rand()%100;
		cout<<ar[i]<<"\t";
	}
	cout<<"\n\n"; cout<<"Enter any
	digit:"; cin>>key;
	ind=BinarySearch(ar,0,SIZE,key);
	cout<<"Index - "<<ind<<"\t";
	cout<<"\n\n";
}
```

İkilik axtarış - çox güclü metoddur. Özünüz mühakimə edin: məsələn, massivin uzunluğu 1023-dür, birinci müqayisədən sonra axtarış sahəsi 511 elementə qədər, ikincidən sonra - 255-ə qədər daralır. Asanlıqla saymaq olar ki, 1023 elementi olan massivdə axtarış üçün 10 müqayisə kifayətdir.

5. Ev tapşırığı

Əfsanəyə görə Hanoyda növbəti konstruksiyanın olduğu məbəd var: 3 almaz mil sancılmışdır, Brahma dünyanı yaradarkən 64 qızıl diski bu millərdən birinə yerləşdirmişdir. Belə ki, ən altda ən böyük disk, onun üzərində nisbətən kiçik disk, ən üstdə isə ən kiçik disk yerləşmişdir. Məbədin kahinləri diskləri növbəti qayda ilə düzməlidirlər:

1. Bir gedişə yalnıız bir diskin yerini dəyişdirmək olar.
2. Böyük diski kiçik diskin üzərinə qoymaq olmaz.

Bu qaydaya əməl edərək kahinlər cari piramidanı 1-ci mildən 3-yə daşımalıdırlar. Onlar bunun öhdəsindən gəldikləri zaman dünyanın sonu olacaqdır.

Biz sizə ev tapşırığı kimi bu məsələni rekursiya ilə həll etmənizi tövsiyə edirik. Müvəffəqiyyətlər!