ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Урок №11

Програмиране на език C

Съдържание

1. Запознаване с рекурсиите

Рекурсия — това е похват в програмирането, при който програмата сама се извиква или непосредствено или косвено.

Неопитния програмист, който за пръв път разбира за рекурсиите би изпитал леко недоумение. Първата му мисъл е, че това е безсмислено! Такава подредба на кода би се превърнала във вечен цикъл, подобен на змия която се е захапала сама себе си, или ще доведе до грешка на етапа за изпълнение, кода на програмата ще погълне всичките ресурси на паметта.

Обаче рекурсията е превъзходен инструмент, който при правилно използване ще помогне на програмиста да реши множество сложни задачи.

Пример за рекурсия

Исторически се получи така, че в качеството на първи пример за рекурсия почти винаги се прилага пример за изчисление на факториел.

Няма да нарушаваме традициите.

Като за начало, да си спомним, какво е факториел. Факториелът се обозначава с удивителен знак „!“ и се изчислява по следния начин:

```
N! = 1 * 2 * 3 * …  * N
```

С други думи, факториела представя произведението на натуралните числа от 1 до N включително. Изхождайки от гореописаната формула, може да се обърне внимание на следващата закономерност:

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

Ура! Можем да намерим факториел чрез самия факториел! Ето тук попадаме в капан. Нашата находка на пръв поглед е абсолютно безполезна, все пак неизвестно понятие се определя чрез същото неизвестно понятие и в резултат получаваме вечен цикъл. Изход от дадената ситуация веднага ще бъде намерен, ако се добави към факториела следващия факт:

```
1!=1
```

Сега ние можем да си позволим да изчислим значението на факториела от всяко число. Да пробваме, на пример да получим 5!, използвайки няколко пъти формулата N! = N * (N–1)! и един път формулата 1! = 1:

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

Как ли ще изглежда дадения алгоритъм ако го пренесем на език С? Нека да пробваме да реализираме рекурсивна функция:

```
#include <iostream>

using namespace std;

long int Fact(long int N)
{
        // ако е проведен опит за изчисление на
      факториел от нула if (N < 1) return 0;
        // ако се изчислява факториел на единица
        // точно тук се провежда изхода от рекурсията
      else if (N == 1) return 1;
   // всяко друго число извиква функцията отново с
      формулата N-1 else return N * Fact(N-1);
}
```

```
void main()
{
   long number=5;
    //първо извикване на рекурсивна
   функция long result=Fact(number);
   cout<<"Result "<<number<<"! is - "<<result<<"\n";
}
```

Както виждате, не е чак толкова сложно. За по-детайлно разбиране на примера, препоръчваме да копирате текста на програмата във Visual Studio и стъпка по стъпка да преминете по кода използвайки дебъгера.

2. Рекурсии или итерации?

Изучавайки предишния раздел на урока – вие най-вероятно сте си задали въпроса: а за какво ни е нужна рекурсията? Все пак да се изчисли факториел може и с помощта на итерации и то съвсем не сложно:

```
#include <iostream>

using namespace std;

long int Fact2(long int N)
{
        long int F = 1;
          //цикъла осъществява пресмятане на
        факториела 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";
}
```

Такъв алгоритъм, вероятно, ще бъде естествен за програмистите. Но, в интерес на истината, това съвсем не е така. Теоретично погледнато, всеки алгоритъм, който може да се реализира рекурсивно, съвсем спокойно може да се реализира и итеративно. Ние току-що се убедихме в това.

Обаче, това съвсем не е така. Рекурсията ще проведе изчислението много по бавно от итерацията. Освен това,. Рекурсията използва много повече оперативна памет в момента на работа.

Означава ли това, че рекурсията е безполезна? В никакъв случай!!! Съществуват много задачи, за които рекурсивното решение е тънко и красиво, а итеративното – сложно, грубо и неестествено. Вашата задача в дадения случай е да се научите не само да оперирате с рекурсии и итерации, но и интуитивно да избирате, кой от подходите да използвате в конкретен случай. От личен опит можем да кажем, че най-добрите места за употреба на рекурсии са задачи, при които е свойствена следната черта: решението на задачата се привежда до решението на същите задачи, но с по-малък размер и следователно, много по лесни за решаване.

Пожелаваме ви успех в тази област! Както се казва: «За да бъдат разбрани рекурсиите, трябва просто да бъдат разбрани рекурсиите!».

3. Бързо сортиране

«Бързото сортиране» е разработено преди около 40 години и е най-широко прилагано и принципно най-ефективния алгоритъм. Метода е основан на разделението на масива на части. Общата схема е следната:

1.От масива се избира опорен елемент a[i].

2.Пуска се функция за разделяне на масива, която премества всички ключове по-малки или равни на a[i] от ляво, а всички ключове по големи или равни на a[i] от дясно, сега масива е съставен от две части, при което елементите от ляво са по малки от елементите от дясно.

3.Ако в подмасива има повече от 2 елемента, рекурсивно поскаме за тях същата функция.

4На края се получава изцяло сортирана последователност.

Да разгледаме масива по детайлно.

Разделяме масива на половина.

Входни данни: масив a[0]...a[N] и елемент p, според когото ще бъде проведено разделянето.

1. Да въведем два указателя: i и j. В началото на алгоритъма те указват, съответно левия и десния край на последователността.

2. Ще предвижваме елемента i със стъпка в 1 елемент по посока края на масива, докато не бъде намерен елемента a[i] >= p.

3. След това по аналогичен начин започваме да движим указателя j от края на масива до началото, докато не бъде намерен a[j] <= p.

4. След това, ако i <= j, разменяме местата на a[i] и a[j] и продължаваме на движим i,j по същите правила.

5. Повтаряме трета стъпка, докато i <= j.

Да разгледаме рисунката къде опорния елемент p = a[3].

Масива се разделя на две части: всички елементи от лявата страна са по-малки или равни на p, всички елементи от дясната са по-големи или равни на p.

Примерна програма.

```
    #include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

template <class T>

void quickSortR(T a[], long N) {
  // На входа - масив a[], a[N] - последния му елемент.
  // да се поставят указатели на изходните места
  long i = 0, j =
  N; T temp, p;
  p = a[ N/2 ];      // централен елемент

  // процедура за
  разделяне 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 );


  // рекурсивно извикване ако има както да се
  сортира 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];
          // преди сортирането
        for(int i=0;i<SIZE;i++){
                ar[i]=rand()%100;
                cout<<ar[i]<<"\t";
        }
        cout<<"\n\n";

        quickSortR(ar,SIZE-1);

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

Алгоритъм на рекурсия.

1. Избиране на опорен елемент p — средата на масива
2. Разделяне на масива спрямо този елемент
3. Ако подмасива от ляво на p съдържа повече от един елемент да се извика quickSortR за него.
4. Ако подмасива дясно от p съдържа повече от един елемент да се извика quickSortR за него.

4. Двоично търсене

В Предишния урок ние разгледахме алгоритъма за линейно търсене, обаче това не е единствената възможност за организирано търсене в масива. Ако имаме масив съдържащ подредена последователност на данни, то в дадения случай е много ефективно двоичното търсене.

Теория на двоичното търсене.
Да предположим, че променливите Lb и Ub съдържат, съответно лявата и дясната граница на отрязък от масива в който се намира необходимия ни елемент. Търсенето винаги ще го започваме с анализ на средния елемент на отрязък от масива. Ако търсеното значение е по малко от средния елемент, ние минаваме към търсене в горната половина на отрязъка, където всички елементи са по-малки от току що проверения. По такъв начин, в резултата на всяка проверка ние смаляваме областта на търсене двойно. Така в нашия пример, след първата итерация в областта на търсене има само три елемента, след втората остава само 1 елемент. По такъв начин, ако дължината на масива е равна на 6, на нас са ни достатъчни три итерации за да намерим нужното число.

```
#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;

          // преди сортирането
        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";
}
```

Двоичното търсене е много мощен метод. Преценете само: например, дължината на масива е равна на 1023, след първото сравнение областта се смалява до 11 елемента а след второто – 255. Лесно е да се сметне, че за търсене в масив от 1023 елемента са достатъчни 10 сравнения.

5. Домашна работа

Легенда разказва, че някъде в Ханой имало храм, в който била разположена следната конструкция: върху основа били поставени 3 елмазени пръта, върху които още при сътворяването на света Брахма нанизал 64 златни диска с отвор по средата, при това най-отдолу се останал най-големият диск, върху него – малко по-малък и така нататък, докато на върха на пирамидата не озовал най-малкият диск.

Жреците в храма трябвало да разместват дисковете съгласно следните правила:

1. За един ход може да се пренесе само един диск.
2. Не може да се слага голям диск върху по-малък.

Спазвайки тези прости правила, жреците трябва да пренесат изходната пирамида от 1-я прът на 3-я. В момента, в който успеят да се справят със задачата, ще настъпи краят на света.

решить данную задачу с помощью рекурсии.

Предлагаме ви за домашна работа да решите дадената задача с помощта на рекурсии. Желаем ви успех!