STEP computer ACADEMY

PROGRAMMING C

Lesson No. 11

Programming

C

Contents

1. Getting acquainted with recursion 3
2. Recursions or iterations? 6
3. Quick sort 8
4. Binary search 11
5. Home assignment. 13

1. Getting acquainted with recursion

Recursion is a programming method implying that the program calls itself directly or indirectly.

As a rule, an amateur programmer after having learnt about the recursion will be taken aback. His first thought would be — it is senseless!!! This chain of calls will turn into an internal loop similar to a snake that has eaten itself; or it will lead to an error at the execution stage, when a program absorbs all memory resources.

Though, a recursion is excellent tool that upon skillful and proper use can help a programmer resolve many difficult tasks.

Example of recursion

Historically it is traditional to take a calculation of factorial in the capacity of the first recursion example.

So we are not going to break the traditions.

At first, let's remember what a factorial is. It is denoted by exclamation point «!» and it is calculated the following way:

```
N! = 1 * 2 * 3 * … * N
```

In other words, factorial represents a product of natural numbers from 1 to N included. Proceeding from the above mentioned formula we can pay attention to the following common pattern:

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

Bingo! We can calculate a factorial through the very factorial! And here we are got trapped. Our discovery, at first sight, is absolutely useless, for an unknown quantity is determined through the same unknown notion, and we get an internal loop. A solution to the situation can be found if we add the following fact to the definition of factorial:

```
1!=1
```

Now we can allow ourselves to calculate the value of factorial of any number. Let's try for example to obtain 5! By means of applying the formula N! = N * (N–1)! Several times and one time we apply the formula 1! = 1::

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

How will look this algorithm if we convert it to the language C? Let's try to realize a recursion function:

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

long int Fact(long int N)
{
        // in case of attempt to calculate factorial of zero
        if (N < 1) return 0;
        // if we calculate a factorial of one
        // here we exercise the exit from recursion
        else if (N == 1) return 1;
   // any other number calls the functions from the beginning with the formula N-1
        else return N * Fact(N-1);
}
```

```
void main()
{
   long number=5;
   //first function call of recursion function
   long result=Fact(number);
   cout<<"Result "<<number<<"! is - "<<result<<"\n";
}
```

As you can see everything is not that difficult. For a more detailed understanding of the example we recommend to copy the program text into Visual Studio and step-by-step pass through the code by a debugger.

2. Recursions or iterations?

Having studied the previous lesson paragraph you might have stumbled on the question: why the recursion is for? In fact, we can realize calculation of factorial using iterations and it is not very difficult to do:

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

long int Fact2(long int N)
{
        long int F = 1;
        //loop exercises factorial calculation
        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";
}
```

Such an algorithm, most probably, will be a natural for the programmers. In reality it is a bit different. From theoretical point of view any algorithm can be realized recursively, and is calmly realized iterative. We just got a confirmation thereof.

Though it is not exactly like that. Recursion makes the calculations slower than iteration. Besides, recursion consumes more RAM upon operation.

Does it mean that the recursion is useless? By no means!!! There is a range of tasks that recursion solution is good and fine, while iterative is difficult, bulky and unnatural. Your

task in this case is to learn not only operate recursion and iteration, but intuitively choose which of the approaches to choose and apply in the particular case. We can add that the best recursion application is resolving the tasks that feature the following characteristic: resolving the task is reduced to resolving the same but smaller in sizes tasks, consequently they are easier of the resolved.

Good luck in this field! As they say: «To understand recursion it is necessary to understand recursion».

3. Quick sort

«Quick sort» was developed around 40 years ago and it is widely used and actually the most efficient algorithm. The method bases on division of an array into parts. General scheme is the following:

1. One controversial element a[i] is chosen out of the array.
2. An array division function is lunched that moves all the keys smaller or equal to a[i] that are on the left of it; or larger or equal to a[i] that are on the right; now the array consists of two parts, wherein elements of the left part are smaller than the right ones.
3. If there are more than two elements in a sub-array we recursively launch the same function for them.
4. In the end we get a completely sorted sequence.

Let's review the algorithm in details.

Divide an array in halves.

Divide an array in halves

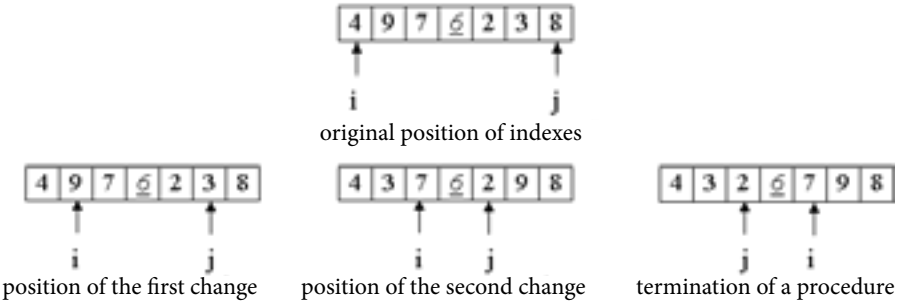
Input data: array a[0]...a[N] and element p according to which the division is to be made.

1. Input two indexes: i and j. In the beginning of the algorithm they indicate to the left and right sequence end respectively.
2. We will move an index i at a pitch of 1 element in the direction towards the array end until an element a[i] >= p is found.
3. Then analogically we move the index j from the array end towards the beginning until a[j] <= p is found.

4. Further if i <= j we interchange a[i] and a[j] and continue moving i,j according to the same rules.

5. Repeat step 3 until i <= j.

Let's review the picture where controversial element p = a[3].



The array has divided into two parts: all elements of the left part are smaller or equal to p, all elements of the right part are larger or equal to p.

Example of a program

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

using namespace std;

template <class T>

void quickSortR(T a[], long N) {
  // at the input - an array a[], a[N] - its last element.
  // put indexes to original places
  long i = 0, j = N;
  T temp, p;

  p = a[ N/2 ];         // central element

  // division procedure
  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 );


  // recursive calls, if there is something to be sorted out
  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];

        // before sort
        for(int i=0;i<SIZE;i++){
                ar[i]=rand()%100;
                cout<<ar[i]<<"\t";
        }
        cout<<"\n\n";
        quickSortR(ar,SIZE-1);

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

Recursion algorithm

1. To choose support element p — array middle
2. To divide an array by this element
3. If a sub-array on the left of p contains more than one element, to call quickSortR for it.
4. If a sub-array on the right of p contains more than one element, to call quickSortR for it.

4. Binary search

In the previous lesson we have reviewed an algorithm of linear search, but this is not the only opportunity to arrange search within array. If there is an array containing an arranged data sequence in this case binary search will be rather effective.

Binary search theory.

Assume that variables Lb and Ub contains respectively left and right boarders of array part containing the element in question. We should always start the search from the analysis of the average element of array part. If a target value is smaller than average element we pass to the search in the upper section of the array part, where all the elements are smaller than just checked one. In other words, (M (average element) – 1) turns into Ub value and at the next iteration we work with the half of an array. Therefore, in the result of each check we narrow the search range twice. In our example the search range covers only three elements after the first iteration and after the second one there is only one element left. Therefore, if an array length is equal to 6 three iterations would be enough to find a desired number.

```
#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;

        // до сортировки
        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";
}
```

Binary search is a powerful method. Judge for yourself: for example an array length is equal to 1023, after first match the search range narrows to 11 elements, after the second one — to 255. It is easy to calculate that for the search within the array out of 1023 elements 10 matches is enough.

5. Home assignment

The legend says that there is a temple somewhere in Hanoi containing the following construction: 3 diamond rods are fixed within the basement; Brahma when creating the world had strung the rods with 64 golden disks with a hole in the middle, wherein the biggest disk happened to be at the bottom, the smaller one on the top of it and so on until the top of the pyramid was finished with the smallest disk. Oracles of the temple should have replaced the disks according to the following rules:

1. Per one move it is allowed to take only one disk.
2. A bigger disk cannot be placed over the smaller one.

Guiding by these simple rules oracles should have carried over an initial pyramid from the first rod to the third one. As soon as they succeed in the task an apocalypses will come.

We offer you as a home assignment to resolve this task using the recursion. Good luck!