Обектно ориентирано програмиране

СТЕК

Стек

Стекът е линейна динамична структура от данни.

Задача. Да се напише програма, която извежда двоичното представяне на естествено число.

Операторът

```
do
{
    cout << k % 2;
    k /= 2;
} while (k);
```

извежда двоичното представяне на числото k, но в обратен ред. За решаване на задачата ще използваме динамичната структура от данни **стек**.

Стек ...

Стекът е крайна редица от елементи от един и същ тип. Операциите включване и изключване на елемент са допустими само за единия край на редицата, който се нарича **връх на стека**. Възможен е достъп само до елемента, намиращ се на върха на стека като достъпът е пряк.

При тази организация на логическите операции, последният включен елемент се изключва пръв. Затова стекът се определя още като структура "*последен влязъл – пръв излязъл*" (LIFO).

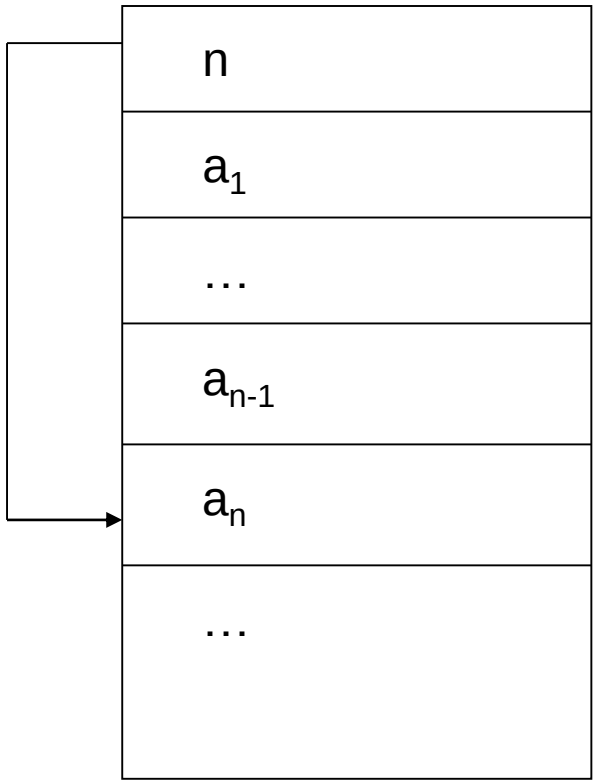
Широко се използват два основни начина за физическо представяне (представяне в ОП) на стек: *последователно* и *свързано*. За целите на тази задача ще използваме последователното представяне.

При това представяне се запазва блок от паметта, вътре в който стекът да расте и да се съкращава. Ако редицата от елементи от един и същ тип

$a_n, a_{n-1}, ..., a_1$

е стек с връх a_n, последователното представяне на стека има вида:

Стек ...



Указател към върха на стека

Връх на стека

Неизползвана част

Стек ...

При включване на елементи в стека, те се поместват в последователни адреси в неизползваната част веднага след върха на стека.

Това физическо представяне ще реализираме като използваме структурата от данни масив. За указател към върха на стека ще служи цяла променлива. Ще използваме подхода абстракция със структури от данни. Първите две нива на абстракция реализираме чрез класа Stack:

Стек ...

```
int const MAX = 100;
class Stack
{
private:
   int arr[MAX]; // масив с елементите на стека
   int top;      // индекс на последния елемент
   // проверка дали стекът е пълен
   bool full() const;
public:
   // Конструктор
   Stack();
```

Стек ...

```
    // функции за достъп
    // проверка дали стек е празен
    bool empty() const;
    // намиране на елемента на върха на стека
    int peek() const;

    // мутатори
    // включване на елемент
    void push(int);
    // изключване на елемент
    int pop();
};
```

Стек ...

Примитивните операции са реализирани чрез следните член-функции:

- `void push(int);` - включва елемент в стека
- `int pop();` - изключва елемент от стека
- `int peek() const;` - намира елемента от върха на стека
- `bool empty() const;` - проверява дали стекът е празен

Стек ...

```
#include "Stack.h"

#include <iostream>
using namespace std;

Stack::Stack()
{
   top = -1; // Празен стек
}

bool Stack::empty() const
{
   return top == -1;
}
```

Стек ...

```
int Stack::peek() const {
   if (empty()) {
      cout << "Грешка: опит за поглеждане в празен стек!\n";
      return 0;
   }
   return arr[top];
}

void Stack::push(int x) {
   if (full()) {
      cout << "Грешка: опит за включване в пълен стек!\n";
   }
   else
      arr[++top] = x;
}
```

Стек ...

```
bool Stack::full() const {
    return top == MAX - 1;
}

int Stack::pop() {
    if (empty()) {
        cout << "Грешка: опит за изключване от празен стек!\n";
        return 0;
    }
    return arr[top--];
}
```

Стек ...

```
/*
* Main.cpp
*/
#include "Stack.h"
#include <iostream>
using namespace std;
void main()
{
   // Прочитане на цяло положително число
   cout << "Number: ";
   int n;
   cin >> n;

   Stack st;
```

Стек ...

```
    int x = n;
    while (x) {
        st.push(x % 2);
        x /= 2;
    }
    // Извеждане на двоичното число
    while (!st.empty()) {
        cout << st.pop();
    }
    cout << endl;
}
```