МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ. ДЕРЕВЬЯ

Методические указания

Воронеж
Издательский дом ВГУ
2015

СОДЕРЖАНИЕ

Рассмотрим реализацию стека на основе массива. Отметим, что кроме непосредственно элементов хранящихся в стеке (для этого используется необходимое количество первых элементов массива), необходимо иметь указатель на вершину стека. Поэтому целесообразно объединить эти данные в один класс. Таким образом, описание стека имеет вид:

```
class Stack
{
 private:
  enum {max = 10};
  int st[max];
  int top;
 public:
  Stack ( ) //конструктор
   {
     this->top = 0;
   }
….
}
```

Отметим, что поле TOP хранит индекс верхнего элемента стека, а одновременно с этим и количество элементов в стеке. Поэтому если TOP==0, то стек пуст.

Схематичное изображение такого представления приведено на рис. 1.2. Вершина стека TOP содержит значение 3, т.е. стек состоит из 3 элементов: 7, 2 и 13. Если из этого стека будет считан элемент, то считается значение 13, а значение TOP сократится до 2. Если же необходимо добавить элемент в стек, то сначала будет увеличено значение TOP, а затем на соответствующее место в массиве будет записан новый элемент.

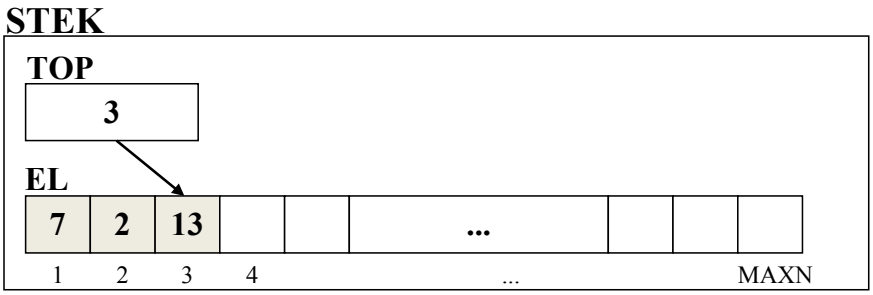

Рис. 1.2. Представление стека с помощью массива

Приведем методы, реализующие основные и дополнительные операции работы со стеком на основе массива.

```
void push ( int var ) //добавить элемент в стек
   {
     st [ ++top ] = var;
   }
  int pop ( ) //извлечь элемент из стека
   {
     return st [ top-- ];
   }
  bool empty() //проверка стека на пустоту
  {
     return top==0;
  }
  void clear ( ) //очистить стек
   {
     top=0;
   }
  void print () //печать содержимого стека
  {
     int s2;
     s2=this->top;
     if (this->empty()==true){
     cout << "Stack is empty.." << endl;
     }
     else{
          cout << "Stack.." << endl;
     while (this->empty()!=true)
          {
               cout << this->pop ( ) << endl;
     }
          this->top=s2;
          }
     }
```

Рассмотрим реализацию стека на основе динамических структур. В этом случае описание стека имеет вид:

```
struct stek
{
int d;
stek *next;//указатель на следующий элемент списка
      // (стека)
};
```

Схематичное представление стека на основе линейного списка приведено на рис. 1.3.

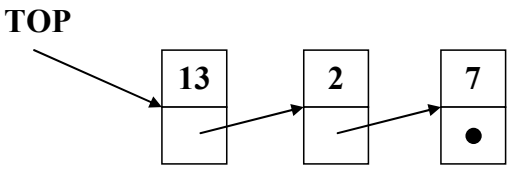

Рис. 1.3. Представление стека с помощью списка

При извлечении элемента из списка, представленного на рис. 1.3, будет получено число 13; указатель TOP будет перенастроен на следующий элемент стека (2); а память, отведенная под хранение элемента 13, будет освобождена. Если же к списку необходимо добавить элемент, то будет создано новое звено списка, в которое будет помещен добавляемый элемент. Затем будет организована ссылка с нового элемента списка на текущую вершину стека, а сам указатель TOP будет перенастроен на новый элемент. Сравнивая рис. 1.2 и рис. 1.3 видно, что в случае реализации стека на основе динамических структур память расходуется существенно эффективнее, т.к. выделяется и освобождается по мере необходимости. Поэтому при решении практических задач обычно используется реализация стека на основе списка [3].

Рассмотрим подпрограммы работы со стеком на основе линейных списков.

```
void init(stek* &st)
{
     st=NULL;
}
bool empty (stek* st)
{
     return (st==NULL);
}
void push(stek* &st, int d)
{
stek *pv = new stek; // объявляем новую динамиче-
           //скую переменную типа stek
pv->d = d; // записываем значение, которое помеща-
      //ется в стек
pv->next = st; // связываем новый элемент стека с
         // предыдущим
st = pv; // новый элемент стека становится его
     //вершиной
}
int pop(stek* &st)
```