24. СТЕКИ

В программировании часто используется структура данных, которая называется **очередью**. Над очередью определены две операции – занесение элемента в очередь и выбор элемента из очереди. При этом выбранный элемент исключается из очереди. В очереди доступны две позиции – ее начало (из этой позиции выбирается элемент из очереди) и конец (в эту позицию помещается заносимый в очередь элемент). Различают два основных вида очередей, отличающихся по дисциплине обслуживания. При первой из дисциплин элемент, поступивший в очередь первым, выбирается первым и удаляется из очереди. Эту дисциплину обслуживания очереди принято называть FIFO (First In – First Out → первый в очередь – первый из очереди).

Остановимся более подробно на очереди с такой дисциплиной обслуживания, при которой на обслуживание первым выбирается тот элемент очереди, который поступил в нее последним. Эту дисциплину обслуживания принято называть LIFO (Last In – First Out → последний в очередь – первый из очереди). Очередь такого вида в программировании называют **стеком** или магазином. В стеке доступна единственная его позиция, называемая вершиной стека. Это позиция, в которой находится последний по времени поступления элемент. Отобразим стек на подходящую структуру данных языка C++.

24.1. Реализация стека через массив

```
// Файл stack0.cpp
typedef char
ETYPE; class stack{
enum{EMPTY = -1};
char* s; int max_len;
int top; public: stack(
){ s = new
ETYPE[100];
max_len = 100; top = EMPTY; }        // Стек
пуст. stack(int size){ s = new ETYPE[size];
max_len = size;
top = EMPTY;
}
stack(const ETYPE a [ ], int len){  // Инициализация массивом.
max_len = len; s = new ETYPE[max_len];
for( int i = 0; i < max_len; i++ ) s[i] = a[i];
```

```
top = max_len - 1;
}
stack(const stack & a){                    // Инициализация стеком.
s = new ETYPE[a.max_len]; max_len
= a.max_len;
top = a.top;
for (int i = 0 ; i < max_len; i++)
s[i] = a.s[i];
}
~ stack(){ delete []s;}
void reset(){ top = EMPTY; }           // Сброс стека в состояние ПУСТ.
void push(ETYPE c ){ s[ ++top ] = c; }          // Занесение в стек.
ETYPE pop (){ return (s[ top-- ]); }               // Извлечение из стека.
ETYPE top_show() const { return (s[top]); } /* Возвращает
элемент из стека, фактически не извлекая его.
```

Модификатор const гарантирует, что эта функция не будет менять данные-члены объектов типа stack

```
*/ int empty() const{return (top == EMPTY);
}
// Проверяет, пуст ли стек. Возвращает
// 1, если стек пуст, 0 – если не пуст.
int full() const{ return (top == max_len - 1); }
// Проверяет, есть ли в стеке еще место.
};
// Конец файла  stack0.cpp
```

Теперь в программе могут появиться такие операторы:

```
stack  data (1000);        // Создание стека длиной 1000.
stack  d[5]                  // Конструктор по умолчанию создает массив
                          // из 5  стеков по 100  элементов каждый.
stack w("ABCD", 4);    // w.s[0] = 'A' . . . s[3] = 'D'. stack cop( w);
    // cop – копия стека w.
```

В качестве примера рассмотрим задачу вывода строки в обратном порядке.

```
# include <iostream> #
include "stack0.cpp"
void main( ){
char str [ ] = "Дядя Вася! ";
stack   s; int i = 0;
cout << str << '\n';
while ( str [ i ] )
if( !s.full ()) s.push(str [ i++]);
else{cout << "Стек заполнен!\n";  break;
} while(!s.empty ())cout << s.pop();   // Печать в обратном порядке.
cout << '\n'; }
```

Результат выполнения программы:

```
Дядя Вася!
!яcaВ ядяД
```

Можно решить эту задачу и так:

```
char str [ ] = "Дядя Вася!";
stack s(str, 10);
while(!s.empty ()) cout << s.pop();  cout
<< '\n';
```

24.2. Реализация стека через динамическую цепочку звеньев

Пусть значением указателя, представляющего стек в целом, является адрес вершины стека. Как и в случае односвязного списка, каждое звено будет содержать указатель на следующей элемент, причем «дно» стека (т. е. элемент, занесенный в стек раньше всех) содержит указатель NULL.

```
// Файл stack.cpp
typedef char
ETYPE; struct elem{
ETYPE data; elem*
next;
elem (ETYPE d, elem* n){ data = d;  next = n; }
}; class stack{ elem* h;                    // Адрес вершины стека.
public: stack (){h = NULL;}                    // Создание пустого
стека. stack(ETYPE a [ ], int  len){ // Инициализация стека
массивом.
h = NULL;
```

```
for (int i = 0; i < len; i++) h = new elem (a[i], h);} stack (stack & a){ //
Инициализация стека другим стеком. elem *p,*q;  p = a.h;  q = NULL;
while (p){ q = new elem (p -> data, q);
if(q->next == NULL) h = q; else  q ->
next -> next = q; p = p -> next;
}
q -> next = NULL;
}
~stack(){reset();}
void push(ETYPE c){h = new elem(c, h);}       // Поместить в стек.
ETYPE pop(){ // Извлечь из стека.
elem* q = h; ETYPE a =  h-> data;
h = h-> next;  delete q;
return  a;
}

ETYPE  pop_show(){return  h -> data;}          // Показать вершину.
void reset(){ while (h ){
elem* q = h;    h = h ->
next;   delete  q;
} } int empty(){ return h ? 0:
1;}
};
// Конец файла stack.cpp
```

Приведем задачу, в решении которой удобно использовать стек.

В файле задана строка литер. Требуется проверить баланс скобок в этой строке.

Баланс соблюдается, если выполнено каждое из следующих условий:

1. Для каждой открывающей скобки справа от нее есть соответствующая закрывающая скобка. Наоборот, для каждой закрывающей скобки слева от нее есть соответствующая открывающая скобка.

2. Соответствующие пары скобок разных типов правильно вложены друг в друга.

Так, в строке {[(a*b) + (n-4)]/[7 - sin(x)] + exp(d)}*s баланс скобок соблюдается, а в строке [{a+b[i]((x - sin(x))]d)) – нет.

В качестве результата работы программы необходимо вывести сообщение о соблюдении баланса. Если баланса нет, то надо вывести начало строки до первого по порядку нарушения баланса скобок.

Для решения задачи сформируем сначала пустой стек. Затем будем последовательно просматривать литеры строки, и открывающие скобки поместим в стек. Если очередной символ окажется закрывающей скобкой, выберем последнюю из занесенных в стек открывающих скобок и сравним эти скобки на соответствие друг другу. Если соответствие есть, то эффект должен быть такой же, как если бы этой пары скобок в строке вообще не было. Если эти скобки не соответствуют друг другу, то не выполнено второе условие соблюдения баланса скобок. Если же в момент выбора из строки очередной закрывающей скобки стек оказался пуст или по завершении просмотра стек оказался не пуст, то не выполнено первое условие соблюдения баланса скобок.

Обозначим через **s** – стек, **sym** – обрабатываемый символ, а через **b** – целую переменную, фиксирующую факт соответствия закрывающей скобки из строки и открывающей скобкой из вершины стека. Для проверки на соответствие закрывающей скобки, являющейся значением переменной **sym**, и открывающей скобки в вершине стека опишем специальную функцию **accord**(), которая возвращает целое значение и удаляет открывающую скобку из стека.

```
# include <fstream> # include
<stdlib.h> # include "stack.cpp"
int accord (stack & s, char sym){
char r = s.pop( ); switch(sym){
case  ')' :       return  r == '('; case
']' :     return  r == '['; case  '}' :
return  r == '{';
default  :        break;
} } void main( ){
ifstream  file( "a.txt" );
if( !file ){
cout << "Ошибка при открытии файла a.txt!\n";  exit(1);
}
stack s; char
sym;
int i, n, b = 1; while( file.peek() !=
EOF && b ){
```

```
file >> sym;   cout
<< sym;
switch(sym){
case '(' : case '[' : case '{' : s.push(sym);  break; case
')' : case ']' : case '}' :
if( s.empty() || !accord (s, sym)) b = 0;  break;
} } if( !b ||
!s.empty())
cout << "\nБаланса скобок нет!\n"; else
cout << "\nСкобки сбалансированы\n";
}
```