# 3. Линейные структуры данных: стек, очередь, дек.

***Цель:*** Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач .

Более подробно материал этого раздела представлен в [2].

## 3.1. Спецификация стека и очереди

Стек, очередь и дек представляют собой разновидности линейного списка.

При задании спецификации линейных списков использовалась модель последовательности [2]. Доступ к каждому элементу последовательности можно получить, продвигаясь по списку от одного элемента к другому. В модели выделяется пройденная часть и текущий элемент, с которого начинается еще не пройденная часть. При этом ранее в курсе лекций и в учебном пособии [2] были определены функции над последовательностями:

* **First** возвращает первый элемент последовательности
* **Last** возвращает последний элемент последовательности
* **Rest** возвращает исходную последовательность, в которой присутствуют все элементы за исключением первого,
* **Lead** возвращает исходную последовательность, в которой присутствуют все элементы за исключением последнего,
* **Prefix** добавляет элемент в начало последовательности
* **Postfix** добавляет элемент в конец последовательности.

Доступ к элементам последовательности (чтение и запись) , осуществляется только через ее начало и конец. Функции First, Rest, Last, Lead − селекторы, а функции Prefix и Postfix − конструкторы.

Если ограничиться только функциями First, Rest, Prefix (или только функциями Last, Lead, Postfix), то получается структура данных, известная как стек (или магазин). Если использовать только набор функций First, Rest, Postfix (или только Last, Lead, Prefix) получим структуру данных, которая называется очередь (англ. queue). Если же используют весь набор функций, то соответствующую структуру данных обычно называют дек (от англ. deq − аббревиатуры сочетания double-ended-queue, т. е. «очередь с двумя концами»). Во все эти структуры данных необходимо добавить еще предикат-индикатор Null, идентифицирующий пустую последовательность, и либо константу, обозначающую пустую последовательность ∆, либо функцию Crtate, порождающую пустую последовательность.

|  |  |
| --- | --- |
| ***Стек Stack*** | *First*, *Rest*, *Prefix* |
| *Last, Lead, Postfix* |
| ***Очередь Queue*** | *First*, *Rest*, *Postfix* |
| *Last, Lead, Prefix* |
| ***Дек Deq*** | *First*, *Rest, Last, Lead, Postfix*, *Prefix* |

Рис. 3.1. Подструктуры последовательности

Исторически сложилось, что при работе со стеком для обозначения функций **First, Rest, Prefix** используют их синонимы **Top, Pop** и **Push** соответственно (Top − верхушка стека, Pop {up} − вытолкнуть (вверх), Push {down} – втолкнуть, вжать). Полезно определить процедуру **Рор2**, совмещающую результат действия функций Top и Pop.

|  |
| --- |
| **procedure** *Pop*2 (**out** *p*: α; **in-out** *s*:  *Stack* (α)); |
| **begin** |
| *p* := *Top* (*s*); *s* := *Pop* (*s*) |
| **end** |

Рис. 3.2. Процедура, совмещающая результат действия функций Top и Pop

## 3.2. Реализация стека и очереди

Ссылочная реализация стека и очереди в динамической памяти в основном аналогична ссылочной реализации линейных списков. Упрощение связано с отсутствием необходимости работать с текущим элементом списка. Идеи такой реализации ясны из рисунка 3.3.

Для ссылочной реализации дека естественно использовать двунаправленный список.

*Стек*:

*Верхушка стека*

*Начало*:

*Конец*:

*Очередь*:

Рис. 3.3. Ссылочное представление стека и очереди

Поскольку для стека, очереди и дека доступ к элементам осуществляется только через начало и конец последовательности, то эти структуры данных допускают и эффективную непрерывную реализацию на базе вектора. При этом используется одномерный массив Mem: array [0..n] of α и переменная Верх :  -1..n.

***Mem:***

***n***

***2***

***1***

***0***

***Верх***

**X X X X X X . . .**

Рис. 3.4. Непрерывная реализация ограниченного стека на базе вектора

Для пустого стека Верх = = -1, для целиком заполненного стека Верх = = n. Вершина стека доступна как Mem [Верх], операция Pop реализуется как Верх:= Верх −1, а операция Push (p, s) как {Верх = Верх + 1; Mem [Верх] = p } при -1 ≤ Верх < n.

Непрерывная реализация ограниченной очереди на базе вектора требует, в отличие от стека, двух переменных Начало и Конец.

***1***

***0***

***2***

***n***

**X X X X X X X**

**X X X X X X**

***Mem:***

***Конец***

***Начало***

Рис. 3.5. Непрерывная реализация ограниченной очереди на базе вектора

Особенностью такого представления является наличие ситуации, когда последовательность элементов очереди по мере их добавления может выходить за границу вектора, продолжаясь с его начала (вектор имитирует здесь так называемый кольцевой буфер). Эта ситуация изображена на рис. 3.6.

***0***

***1***

***2***

***n***

***Mem:***

**X X X X X X X X X X X**

***Начало***

***Конец***

Рис. 3.6. Непрерывное представление очереди в кольцевом буфере

Переменная **Начало** может принимать значение **Конец+1** в случаях как пустой, так и полной очереди. Чтобы различать эти ситуации, надо ввести еще одну переменную **Длина**. Для пустой очереди Длина = 0, а для полной очереди Длина = n+1.

Так же может быть реализован и дек.

## 3.3. Рекомендации по изучению раздела и выполнению индивидуального задания

При изучении данного раздела студент должен ознакомиться с материалом раздела 2 пособия [2], выбрать индивидуальное задание из списка заданий, указанных в **Приложении 3**. Каждое задание предполагает самостоятельную разработку студентом одного или нескольких модулей на языке С++, реализующих тот или иной вариант стека, очереди или дека, а также главной программы, непосредственно решающей поставленную задачу.

В заданиях 1, 2, 3 следует использовать очередь и операции над ней. В заданиях 4 – 8 следует использовать стек и операции над ним. В заданиях с 9 по 11 следует использовать очередь и/или стек и операции над ними. При этом и стек и очередь могут быть реализован как на базе вектора, так и в связанной памяти (ссылочная реализация).

## 3.4. Пример реализации и использования стека для вычисления арифметического выражения в постфиксной форме

Напишем программу вычисления арифметического выражения в постфиксной форме. Суть работы программы состоит в следующем. В стек последовательно записываются символы входной строки, представляющей собой арифметическое выражение в постфиксной форме. Последовательно идущие друг за другом цифры преобразуются в числа. При появлении знака операции из стека последовательно выталкиваются 2 операнда, над ними выполняется операция, соответствующая этому знаку, результат помещается в стек и чтение входной строки продолжается.

Ссылочная реализация стека в динамической (связанной) памяти допускает несколько вариантов.

**Вариант 1:** Используется не шаблонный класс. Интерфейс и реализация стека разделены: st\_interface.h － файл интерфейса стека, st\_implementation.cpp － файл реализации стека. Файл интерфейса подключается к главной программе main1.cpp директивой #include “st\_interf1.h”, а также к модулю реализации.

// Программа клиент вычисляет арифметическое выражение, заданное в постфиксной форме

// Ссылочная реализация в динамической (связанной) памяти

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <windows.h>

#include "st\_interf1.h"

using namespace std;

using namespace st\_modul1;

int main () {

char a[100];

// это вставка для правильной кодировки русских букв

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//

cout << "ввод строки с постфиксной записью выражения" << endl;

ifstream fin("postfix.txt");

fin >> noskipws; // включить манипулятор!

if (!fin ) {cout << "File not open for reading!\n"; return 1;}

int n1 = 100;

int n = 0;

while (n<n1 && fin >> a[n]) n++;

//вывод строки

cout << "длина строки = " << n << endl;

for (int i=0; i<n; i++) cout << a[i];

cout << endl;

cout << "вычислить!" << endl;

Stack s;

for (int i = 0; i < n; i++)

{ cout << "шаг: " << i+1 << " символ = " << a[i] << endl; // dem

if (a[i] == '+')

s.push(s.pop2() + s.pop2());

if (a[i] == '\*')

s.push(s.pop2() \* s.pop2());

if ((a[i] >= '0') && (a[i] <= '9'))

s.push(0);

while ((a[i] >= '0') && (a[i] <= '9'))

{s.push(10\*s.pop2() + (a[i++]-'0'));

cout << "шаг\_: " << i+1 << " символ = " << a[i] << endl;// dem

}

}

cout << "Результат =" << s.pop2() << endl;

s.destroy();

return (0);

}

// интерфейс АТД "Стек" (ссылочная реализация в динамической памяти)

namespace st\_modul1

{

//-------------------------------------

typedef int base;

class Stack {

private:

struct node;

/\* определение структуры будет дано в другом файле (продолжении namespace st\_modul) － в файле Implementation,

а здесь достаточно объявления "struct node;"

\*/

node \*topOfStack;

public:

Stack ()

{ topOfStack = NULL;

};

base top (void);

void pop (void);

base pop2(void);

void push (const base &x);

bool isNull(void);

void destroy (void);

};

}

// Implementation － Реализация АТД "Стек"(ссылочная реализация в динамической памяти)

#include <iostream>

#include <cstdlib>

#include "st\_interf1.h"

using namespace std ;

namespace st\_modul1

{

struct Stack::node {//

base \*hd;

node \*tl;

// constructor

node ()

{hd = NULL; tl = NULL;

}

};// end node

//-------------------------------------

base Stack::top (void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: top(null) \n"; exit(1); }

else return \*topOfStack->hd;

}

//-------------------------------------

void Stack::pop (void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{ node \*oldTop = topOfStack;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

}

}

//-------------------------------------

base Stack::pop2(void)

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{ node \*oldTop = topOfStack;

base r = \*topOfStack->hd;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

return r;

}

}

//-------------------------------------

void Stack::push (const base &x)

{ node \*p;

p = topOfStack;

topOfStack = new node;

if ( topOfStack != NULL) {

topOfStack->hd = new base;

\*topOfStack->hd = x;

cout << "push -> " << x << endl; // Demo

topOfStack->tl = p;

}

else {cerr << "Memory not enough\n"; exit(1);}

}

//-------------------------------------

bool Stack::isNull(void)

{ return (topOfStack == NULL) ;

}

//-------------------------------------

void Stack::destroy (void)

{ while ( topOfStack != NULL) {

pop();

}

}

} // end of namespace st\_modul1

**Вариант 2.** Используется шаблонный класс, а интерфейс, и реализация располагаются в одном заголовочном файле st\_interf2.h.

// Интерфейс АТД "Стек" (ссылочная реализация в динамической памяти)с шаблоном класса

// И интерфейс, и реализация в одном заголовочном файле

namespace st\_modul2

{

//-------------------------------------

template <class Elem>

class Stack

{

private:

struct node

{ //

Elem \*hd;

node \*tl;

// constructor

node ()

{hd = NULL; tl = NULL;

};

};// end node

node \*topOfStack;

public:

Stack ()

{ topOfStack = NULL;

}//;

//-------------------------------------

Elem Stack::top (void) //

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: top(null) \n"; exit(1); }

else return \*topOfStack->hd;

}

//-------------------------------------

void Stack::pop (void)//

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{ node \*oldTop = topOfStack;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

}

}

//-------------------------------------

Elem Stack::pop2(void)//

{// PreCondition: not null

if (topOfStack == NULL) { cerr << "Error: pop(null) \n"; exit(1); }

else

{ node \*oldTop = topOfStack;

Elem r = \*topOfStack->hd;

topOfStack = topOfStack->tl;

delete oldTop->hd;

delete oldTop;

return r;

}

}

//-------------------------------------

void Stack::push (const Elem &x)//

{ node \*p;

p = topOfStack;

topOfStack = new node;

if ( topOfStack != NULL) {

topOfStack->hd = new Elem;

\*topOfStack->hd = x;

cout << "push -> " << x << endl; // Demo

topOfStack->tl = p;

}

else {cerr << "Memory not enough\n"; exit(1);}

}

//-------------------------------------

bool Stack::isNull(void)//

{ return (topOfStack == NULL) ;

}

//-------------------------------------

void Stack::destroy (void)//

{ while ( topOfStack != NULL) {

pop();

}

}

};

}

Главная программа при этом не меняется по сравнению с первым вариантом, если не считать, что директива #include “st\_interf1.h” изменится на #include “st\_interf2.h”, а Stack s на Stack<int> s.

**Вариант 3.** Используется шаблонный класс. Интерфейс и реализация располагаются в разных h- файлах. Файл интерфейса подключается к главной программе, а файл реализации к заголовочному файлу (файлу интерфейса). Студенты могут предложить такой вариант реалиации самостоятельно.

Рассмотрим теперь непрерывную реализацию ограниченного стека на базе вектора.

Интерфейс и реализация будут располагаться в одном заголовочном файле «st\_templ.h», используем шаблонный класс. Главную программу вычисления выражения в постфиксной форме назовем «postfix\_calc.cpp».

// Программа клиент вычисляет арифметическое выражение, заданное в постфиксной форме

// Непрерывная реализация на базе вектора

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <windows.h>

#include "st\_templ.h"

using namespace std;

int main ( )

{ char a[100]; //

int maxN;

// это вставка для правильной кодировки русских букв

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//

cout << "введите максимальный размер стека =" << endl;

cin >> maxN;

cout << "введен максимальный размер стека =" << maxN << endl;

cout << "ввод строки с постфиксной записью выражения" << endl;

ifstream fin("postfix.txt");

fin >> noskipws; // включить манипулятор!

if (!fin ) {cout << "File not open for reading!\n"; return 1;}

int n1 = 100;

int n = 0;

while (n<n1 && fin >> a[n]) n++;

//вывод строки

cout << "длина строки = " << n << endl;

for (int i=0; i<n; i++) cout << a[i];

cout << endl;

cout << "вычислить!" << endl;

STACK<int> save(maxN);

for (int i = 0; i < n; i++)

{ cout << "шаг " << i+1 << " символ = " << a[i] << endl;

if (a[i] == '+')

save.push(save.pop() + save.pop());

if (a[i] == '\*')

save.push(save.pop() \* save.pop());

if ((a[i] >= '0') && (a[i] <= '9'))

save.push(0);

while ((a[i] >= '0') && (a[i] <= '9'))

{save.push(10\*save.pop() + (a[i++]-'0'));

cout << "шаг\_: " << i+1 << " символ = " << a[i] << endl;

}

}

cout << save.pop() << endl;

}

// интерфейс АТД "Стек"

// шаблонный класс. Без некоторых проверок...

// и интерфейс, и реализация в этом заголовочном файле

template <class Item>

class STACK

{

private:

Item \*s; int N; int N1; // my N1

public:

STACK(int maxN)

{ s = new Item[maxN]; N = 0; N1 = maxN;} // my N1

int empty() const

{ return N == 0; }

void push(Item elem)

{ s[N++] = elem;

if (N > N1) cout << "ошибка!!!" << endl; // my N1

}

Item pop()

{ return s[--N]; }

};

///}