Алгоритмы и структуры данных

ТЕМА #3:

СТЕК: ИСПОЛЬЗОВАНИЕ, РЕАЛИЗАЦИЯ

Стек: использование, реализация

1 Формы записи арифметических выражений

2 Алгоритм получения обратной польской записи.

3. Алгоритм вычисления значения выражения по обратной польской записи

4. Структура данных: Стек. Реализация.

1. Формы записи арифметических выражений

- ✓ Арифметические выражения, состоящие только из констант, бинарных операций и круглых скобок, определяющих последовательность выполнения действий.

- ✓ При отсутствии скобок последовательность действий определяется стандартными соглашениями о приоритете операций**.** *Рассматриваем операции, выполняющиеся слева направо (исключаем возведение в степень).*

1. Формы записи арифметических выражений

Инфиксная

Префиксная

Постфиксная

При записи арифметических выражений в привычной (естественной) форме бинарные операции записываются между своими операндами.

Например,

1+2

1*3+5

(1+2)/3

Это пример инфиксной формы записи.

Рассмотрим бесскобочные линейные формы записи арифметических выражений.

“Забудем” о приоритетах операций и унарных операциях,

установим порядок вычисления только с помощью скобок.

Тогда арифметическое выражение

4*(6-3)+(8-6)/2

можно представить в форме

(4*(6-3))+((8-6)/2)

Добавим к формуле “лишние” скобки и получим то же выражение в расширенной скобочной форме:

((4*(6-3)) + ((8-6)/2))

((4*(6-3)) + ((8-6)/2))

А теперь начнем изменять запись этого выражения.

Внешнее действие **(A+B)** запишем, снимая его скобки, в виде **+ A B**

или, для нашего, примера

+ (4* (6-3)) ((8-6)/2)

Аналогично преобразуем аргумент А

+ * 4 (6-3) ((8-6)/2)

А дальше будем преобразовывать каждую следующую скобку, двигаясь от внешних скобок к внутренним:

+ * 4 - 6 3 ((8-6)/2)

+ * 4 - 6 3 / (8-6) 2

+ *4 - 6 3 / - 8 6 2

+ *4 - 6 3 / - 8 6 2

Получившаяся форма называется префиксной записью (действие пишется перед операндами) или польской записью, так как ее появление связывается с именем польского математика Яна Лукашевича (Jan Łukasiewicz, 1878-1956).

Алгоритм вычисления выражения, записанного в префиксной форме:

- читается первый элемент записи
- если это число, то оно является результатом,
- если это знак действия, то из остатка записи берутся два числа (аналогичным образом), и с ними производится нужное действие.

1. Формы записи арифметических выражений

result (+ * 4 - 6 3 / - 8 6 2) = A1 + B1,

где A1 и B1 нужно еще вычислить из записи:

* 4 - 6 3 / - 8 6 2

Далее

result (* 4 - 6 3 / - 8 6 2) = A2 * B2,

Опять таки, A2 = 4, а B2 вычисляется из оставшейся записи:

result (- 6 3 / - 8 6 2) = A3 - B3

Тут уже просто: A3 = 6, B3 = 3, так что B2 = 3, и мы нашли A1 = 3*4 = 12.

Настала очередь операнда B1, который находится по остатку записи:

result (/ - 8 6 2) = A4 / B4

и так далее.

Наибольшее распространение получил метод компиляции с помощью обратной польской записи, которую также предложил польский математик Я. Лукашевич.

Обратная польская запись, иначе называемая постфиксной записью, предполагает запись операции справа после своих операндов.

Приведенные выше выражения записаны в постфиксной форме:

12+

13*5+

12+3/

4 6 3 - * 8 6 - 2 / +

Алгоритм вычисления инфиксного выражения:

1. Преобразовать инфиксное выражение в постфиксную форму
2. Вычислить значение по полученной постфиксной форме.

Каждый из этих алгоритмов требует одного “прохода” выражения слева направо и использует стек для необходимых операций.

2. Алгоритм получения обратной польской записи

Вычисление выражения, записанного в обратной польской записи, может проводиться путем однократного просмотра, что является весьма удобным при генерации объектного кода программ.

Получение обратной польской записи из исходного выражения осуществляется на основе простого алгоритма, предложенного Дейкстрой

Приоритет операции:

операция	*приоритет*
(	0
)	1
+ -	2
* /	3
^	4

Алгоритм получения обратной польской записи

Просматриваем исходную строку символов слева направо.

Операнды переписываем в выходную строку, а знаки бинарных операций заносим в стек по следующему правилу:

1) если стек пуст, то операция из входной строки заносится в стек;
2) если стек не пуст, операция выталкивает из стека все операции с большим или равным приоритетом в выходную строку;
3) если очередной символ из исходной строки - открывающая скобка, то он проталкивается в стек;
4) закрывающая скобка выталкивает все операции из стека до ближайшей открывающей скобки, сами скобки в выходную строку не переписываются, а уничтожают друг друга;
5) В завершении оставшиеся операции из стека переносятся в выходную строку.

Пример получения обратной польской записи выражения **(A+B)*(C+D)-E .**

Номер символа	1	2	3	4	5	6	7	8	9	10	11	12	13	
Входная строка	(	A	+	B	)	*	(	C	+	D	)	-	E	
Состояние стека	(	(	+ (	+ (		*	(*	(*	+ (*	+ (*	*	-	-	
Выходная строка		A		B	+			C		D	+	*	E	-

3. Алгоритм вычисления значения выражения по обратной польской записи

Считаем, что

- каждое поступившее число просто «заталкивается» в стек
- каждая операция снимает со стека два числа, выполняет с ними свое действие и «заталкивает» результат в стек

3. Алгоритм вычисления значения выражения по обратной польской записи

Пример.

4 6 3 - * 8 6 - 2 / +

Входной символ	Содержимое стека
4	4
6	4 6
3	4 6 3
-	4 3
*	12
8	12 8
6	12 8 6
-	12 2
2	12 2 2
/	12 1
+	13

3. Алгоритм вычисления значения выражения по обратной польской записи

Вместо числовых значений можно ставить имена переменных и строить бесскобочные алгебраические выражения.

Можно также использовать имена функций, которые должны снимать со стека нужное число аргументов и записывать в стек нужное число результатов.

3. Алгоритм вычисления значения выражения по обратной польской записи

Алгоритм обработки входного постфиксного выражения (корректного, не пустого!):

Значения операндов и операций представлены одиночными символами, причем все операции двухместные, операции одного приоритета выполняются слева направо (нет возведения в степень).

1) Подготовить пустой стек.
2) Если очередной элемент потока - операнд (число), то поместить его в стек и перейти к пункту 2
3) Если очередной элемент потока – операция (двухместная), вытащить из стека, по очереди, два верхних элемента (это числа); рассматривая их, соответственно, как 2-й и 1-й операнды двухместного выражения, вычислить его значение; поместить его в стек; перейти к пункту 2.
4) Если входной поток пуст, то вытащить элемент из стека, вывести его значение в качестве результата и закончить обработку.

Если допустить во входном потоке операнды и операции, "не помещающиеся" в один символ, то механизм, описанный в алгоритме, изменится мало.

В этом случае придется анализировать в качестве "претендента на звание" операнда или операции не по одному символу, а накапливать предварительно подстроки.

4. Структура данных: стек

АТД стек – Stack – абстрактный класс

СД стек «ограниченный» - StackArray – реализован через массив

СД стек «неограниченный» - StackList – реализован через список (односвязный)

СД стек «неограниченный» - StackVector – реализован через Vector ?

Шаблонные классы

Класс Stack – абстрактный

```
template <class T>
class Stack
{
public:
   virtual ~Stack() {}                    //
   virtual void push(const T& e) = 0; // Добавление элемента в стек
   virtual T pop() = 0;                   // Удаление и возвращение верхнего элемента.
                                          // Если элементов нет, может возникнуть StackUnderflow
   virtual bool isEmpty() = 0;            // Проверка стека на пустоту

};
```

```
virtual const T& top() = 0;          // Возвращение верхнего элемента (без его удаления из стека)
```

Класс Stack – абстрактный

Исключительные ситуации:

```
class Stack
      exception: StackUnderflow

class StackArray: public Stack …
      exception: StackUnderflow, StackOverflow, WrongStackSize

class StackList: public Stack
      exception: StackUnderflow

class StackVector: public Stack
      exception: StackUnderflow
```

```
template <class T>
class StackArray : public Stack<T>
{
public:
   StackArray(size_t size = 100);                    // size задает размер стека "по умолчанию"
   StackArray(const StackArray<T>& src);             // = delete;
   StackArray(StackArray<T>&& src);
   StackArray& operator=(const StackArray<T>& src); // = delete;
   StackArray& operator=(StackArray<T>&& src);
   virtual ~StackArray();                            //{ delete[] array_; }
   void push(const T& e);
   T pop();
   bool isEmpty();                                   // { return top_ == 0; }
private:
   T*     array_;      // массив элементов стека: !!! array_[0] – не используется, top_ от 1 до size_
   size_t top_ ;       // вершина стека, элемент занесенный в стек последним
   size_t size_;       // размер стека
   void swap(StackArray<T>& src);
};
```

```
template <class T>
void StackArray<T> ::swap(StackArray<T>& src)
{
   std::swap (array_, src.array_);      // (this->array_, src.array_)
   std::swap (top_,   src.top_);
   std::swap (size_,  src.size_);
}
```

```
template <class T>
StackArray<T>::StackArray(size_t size) :
    size_(size),
    top_(0)
{ // !!! array_[0] – не используется, top_ от 1 до size_
  try {
     array_ = new T[size + 1]; // пытаемся заказать память под элементы стека...
  }
  catch (...) {                // если что-то случилось (например, размер слишком большой
     throw WrongStackSize();   // или отрицательный) - возникает исключительная ситуация
  }
}
```

```
template <class T>
void StackArray<T>::push(const T& e)
{
    if (top_ == size_) {
        throw StackOverflow(); // нет места для нового элемента
    }
    array_[++top_] = e;            // занесение элемента в стек
}
```

```
template <class T>
const T& StackArray<T>::pop()
{
   if  (isEmpty()) {        // (top_ == 0)
     throw StackUnderflow(); // стек пуст
   }
   return array_[top_--];  // Элемент физически остается в стеке, но больше "не доступен"
}
```

Обработка исключений:

Использование класса exception из файла <exception>

Создание своего класса для обработки исключений

Обработка исключений: Использование класса exception из файла <exception>

```
#include < exception >

// Классы StackOverflow и StackUnderflow представляют две основные

// исключительные ситуации, которые могут возникнуть при работе со стеком

class StackOverflow : public std::exception

{

public:

        StackOverflow(): reason_("Stack Overflow") {}

        const std::string& what() const { return reason; }

private:

        const std::string reason_;  // ! const

};
```

Обработка исключений: Использование класса exception из файла <exception>

```
class StackUnderflow : public std::exception
{
public:
        StackUnderflow() : reason_("Stack Underflow") {}
        const std::string& what() const { return reason; }
private:
        const std::string& reason_;  // ! const


};
```

Обработка исключений: Создание своего класса для обработки исключений

```
// в шаблоне базового класса -
public:
    // базовый класс исключения
    class StackException
    {
    public:
        StackException(const std::string &);
        std::string getMessage() const;
    private:
        std::string message_;
    };
```

Обработка исключений: Создание своего класса для обработки исключений

```
// исключение, возникающее при извлечении значения из пустого стека
    class StackUnderflow : public StackException
    {
    public:
         StackUnderflow();
    };
// исключение, возникающее при переполнении стека
    class StackOverflow() : public StackException
    {
    public:
        StackOverflow();
    };
```

```
StackArray:: StackOverflow (const std::string & message) :
    message_(message)
{}

std::string StackArray:: StackOverflow ::getMessage() const
{
    return message_;
}
```

В обработчике исключений:

```
catch (const StackArray::StackOverflow & e) {
     std::cerr << e.getMessage();
}
```

```
// Конструктор копирования – создание копии имеющегося списка, если не указано delete
template <class T>
StackArray<T>::StackArray(const StackArray<T>& src):
    size_ (src.size_),
    top_( src.top_)
{
   try {// !!! array_[0] – не используется, top_ от 1 до size_
      array_ = new T[src.size_+1];
   }
   catch (...) {
       throw WrongStackSize();
   }
   // копирование массива
   for (size_t i = 1; i < src.top_; i++) {
        array_[i] = src.array_[i];
   }
}
```