

Абстрактный тип данных стек

Примеры
использования
абстрактного стека

Абстрактный тип данных Стек

- *Стеком* называется последовательность элементов одного и того же типа, к которой можно добавлять новые элементы и удалять элементы последовательности.

Причем как добавление элементов, так и удаление элементов производится с одного и того же конца последовательности, называемого *вершиной стека*.

Операции со стеком

- Создание пустого стека
- Уничтожение стека
- Определение пуст ли стек
- Добавление нового элемента в стек
- Удаление верхнего элемента из стека
- Извлечение из стека значения верхнего элемента (вершины стека) без его удаления

Диаграмма абстрактного стека



Операции со стеком

- ***createStack()*** - создает пустой стек
- ***destroyStack ()***– уничтожает стек
- ***isEmpty()*** – функция определения пустоты стека ли стек
- ***push(NewElement)*** – добавляет новый элемент *NewElement* в стек
- ***pop()*** – удаляет верхний элемент из стека
- ***getTop()***– возвращает значение верхнего элемента (вершины стека) без его удаления

СТЕК

A diagram showing the word 'СТЕК' at the top, with two arrows pointing downwards to the left and right. Below the arrows is a table with two columns and two rows. The left column is under the arrow pointing left, and the right column is under the arrow pointing right. The table compares 'Ограниченный' (Limited) and 'Неограниченный' (Unlimited) stacks.

Ограниченный	Неограниченный
Количество элементов ограничивается некоторым числом	Размер ограничен только доступной памятью
Реализуется с помощью массива	Реализуется в виде связного списка или в виде абстрактного списка

Реализация ограниченного стека в виде массива

- Размер массива определяет максимальное число элементов в стеке
- Необходимо определить указатель *top* положения верхнего элемента
- При вставке элемента производится увеличение значения *top* на единицу
- При удалении элемента производится уменьшение значения *top* на единицу

Реализация ограниченного стека в виде массива

Пусть `TypeItem` – тип элементов стека

`Max_size` – максимальный размер стека

`Stack [Max_size]` – массив элементов стека

`top` - указатель на вершину стека

Основные операции со стеком

```
void createStack() { top=0;}  
void pop()  
{if ( top==0) cout<<'Стек пуст';  
  else  --top;} //конец функции pop  
void push(TypeItem NewItem)  
{ if (top>=Max_size) cout<<'Стек полон';  
  else Stack[++top]=NewItem } //конец  
  //функции push
```

Основные операции со стеком

```
TypeItem getTop()  
{if (top==0) cout<<'Стек пуст';  
  else  
  return (Stack[top];  
}  
int sEmpty()_{return(top==0);}  
  
void_destroyStack () { top=0;}
```

Ограниченный стек

- Еще одной реализацией ограниченного стека является описание стека с помощью динамического массива
- Достоинством этого метода является возможность выделения памяти по мере необходимости

Ограниченный стек

Struct Stack

```
{ TypeItem *array;  
  int count,max_size;  
}
```

Реализация стека в виде СВЯЗНОГО СПИСКА

Пусть `StackItemType` – тип элементов стека

`// Узел стека`

```
Struct StackNode
```

```
{
```

```
    StackItemType Item;
```

```
    StackNode * next;
```

```
};
```

```
StackNode *top; // указатель на первый элемент
```

Реализация стека в виде связанного списка

```
class Stack
```

```
{
```

```
    public:
```

```
    // Конструкторы и деструкторы:
```

```
    // Конструктор по умолчанию
```

```
    Stack();
```

```
    // Конструктор копирования
```

```
    Stack(const Stack& aStack) ; //
```

```
    Деструктор
```

```
    ~Stack();
```

Реализация стека в виде связанного списка

// Операции над стеком:

int isEmpty() const;

void push(StackItemType newItem)

void pop()

void pop(StackItemType& stackTop)

void getTop(StackItemType&
stackTop) const

Реализация стека в виде СВЯЗНОГО СПИСКА

private:

```
struct StackNode // Узел стека
{
StackItemType item; //Данные,
содержащиеся //в узле стека
StackNode *next; // Указатель на
следующий узел
}; // end struct

StackNode *top; // Указатель на первый
элемент стека
}; // Конец класса Stack
```


Конструкторы и деструкторы:

```
Stack::Stack() : top(NULL)
```

```
{ } // Конец Конструктора по умолчанию
```

```
Stack::Stack(const Stack& aStack)
```

```
{ //первый узел
```

```
    if (aStack.top == NULL) top=NULL;
```

```
    else {
```

```
        top = new StackNode;
```

```
        top->item = aStack.top->item; }
```

Конструкторы и деструкторы:

```
//остальная часть списка
```

```
StackNode *newPtr=top;
```

```
for(StackNode *cur= aStack.top->next;
```

```
    cur!=NULL;  cur=cur->next)
```

```
{ newPtr->next=new StackNode;
```

```
  newPtr=newPtr->next;
```

```
  newPtr->item=cur->item;
```

```
}
```

```
newPtr->next=NULL;
```

```
}
```

```
} // Конец Конструктора копирования
```

Конструкторы и деструкторы:

```
Stack::~~Stack()
```

```
{ while(!isEmpty()) pop(); }
```

Операции со стеком

```
Stack::pop()
{ if (isEmpty()) cout<<"стек пуст";
  else
  {
    StackNode *temp=top;
    top=top->next;
    temp->next=NULL;
    delete temp;
  }
} // Конец функции pop
```

Операции со стеком

```
Stack::pop(StackItemType & stackTop)
{ if (isEmpty()) cout<<"стек пуст";
  else
  {
    stackTop=top->item;
    StackNode *temp=top;
    top=top->next;
    temp->next=NULL;
    delete temp;
  }
} // Конец функции pop
```

Операции со стеком

```
Stack::push(StackItemType newItem)
```

```
{  
    StackNode *temp=new StackNode;  
    temp->item=newItem;  
    temp->next=top;  
    top=temp  
} // Конец функции push
```

Операции со стеком

```
int Stack::isEmpty()
```

```
{  
    return (top==NULL)  
} // Конец функции isEmpty
```

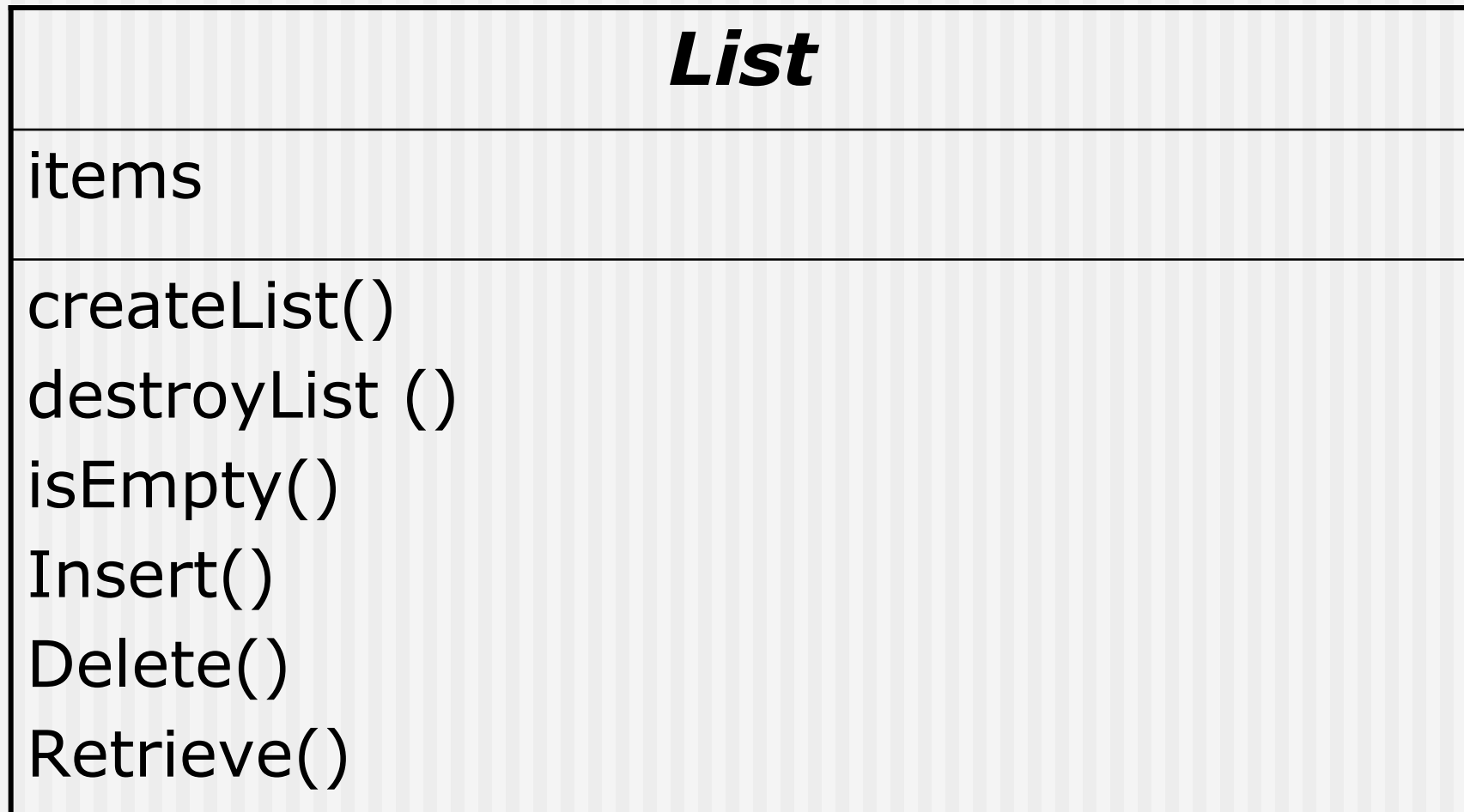
```
void Stack::getTop(StackItemType & stackTop)
```

```
{  
    if(isEmpty) cout<<`стек пуст`;  
    else stackTop=top->item;  
} // Конец функции getTop
```

Реализация стека в виде абстрактного списка

```
class Stack
{
    public:
        //Конструкторы и деструкторы
        .....
        // Операции над стеком
        .....
    private:
        List L; // список элементов стека
} // конец описания класса
```


Реализация стека в виде абстрактного списка Диаграмма класса Список



Реализация стека в виде абстрактного списка

```
Stack::Stack(const Stack& aStack):  
    L(aStack.L)  
{  
};
```

```
int Stack::isEmpty() const  
{  
    return(L.isEmpty);  
}
```

Реализация стека в виде абстрактного списка

```
Stack::push(StackItemType newItem)
{
    L.insert(1,newItem);
};
```

```
void Stack::pop()
{
    L.remove(1);
}
```

Реализация стека в виде абстрактного списка

```
Stack::getTop(StackItemType& stackTop)
{
    L.retrieve(1,stackTop);
};
```

```
void Stack::pop(StackItemType& stackTop)
{ L.retrieve(1,stackTop);
  L.remove(1);
}
```

Алгебраические выражения

- *Инфиксная запись выражений:*
каждый бинарный оператор помещается между своими операндами
- *Префиксная запись выражений (Prefix):*
каждый бинарный оператор помещается перед своими операндами
(Польская запись)
- *Постфиксная запись выражений (Postfix):*
каждый бинарный оператор помещается после своих операндов
(Обратная Польская запись)

Алгебраические выражения

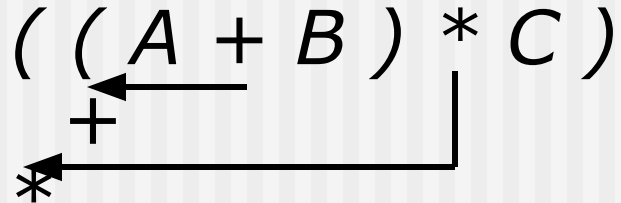
<i>Инфиксная форма</i>	<i>Префиксная форма</i>	<i>Постфиксная форма</i>
$A+B$	$+AB$	$AB+$
$A+B*C$	$+A*BC$	$ABC*+$
$(A+B)*C$	$*+ABC$	$AB+C*$

Преобразование инфиксной формы в Prefix и Postfix

- Допустим, инфиксное выражение полностью заключено в скобки
- Преобразование в префиксную форму: каждый оператор перемещается на позицию соответствующей открывающейся скобки (перед операцией)
- Преобразование в постфиксную форму: каждый оператор перемещается на позицию соответствующей закрывающейся скобки (после операцией)
- Скобки убираются

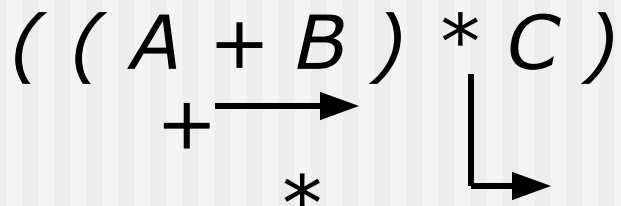
Примеры

- Преобразование в префиксную форму:

$$((A + B) * C)$$


$$*+ABC$$

- Преобразование в постфиксную форму:

$$((A + B) * C)$$


$$AB+C*$$

Преимущества префиксной и постфиксной форм записи

- Не нужны приоритеты операций, правила ассоциативности, скобки
- Алгоритмы распознавания выражений и вычисления более просты

Вычисление постфиксных выражений

- Допустим необходимо вычислить выражение:
 $2*(3+4)$
- Его постфиксная запись:
 $234+*$
- Порядок выполнения операций:
 - Помещаем в стек значения всех операндов, пока не встретим знак операции
 - Выталкиваем из стека 2 операнда
 - Производим с ними соответствующую операцию
 - Результат помещаем в стек
 - Повторяем операции до тех пор, пока не кончится строка символов

Пример (Функция $Pop(op)$ - возвращает значение op из вершины стека и удаляет это значение из стека):

Символ	Действия	Состояние стека
2	Push(2)	2
3	Push(3)	2 3
4	Push(4)	2 3 4
+	Pop(Op2)	2 3
	Pop(Op1)	2
	Result=Op1+Op2	2
	Push(Result)	2 7
*	Pop(Op2)	2
	Pop(Op1)	-
	Result=Op1*Op2	-
	Push(Result)	14

Псевдокод алгоритма

Предположения:

- Строка содержит синтаксически правильное выражение
- Унарные операции и операции возведения в степень не используются
- Операнды задаются строчными буквами

Псевдокод алгоритма

```
For (каждый символ ch в строке)
{ if (ch является операндом)
  // помещаем ch в стек
  Push(ch);
else
{
  Oprgn=ch;
  Op2= getTop();Pop(); //извлекаем значение из вершины
  Op1= getTop(); Pop(); // и удаляем его
  // выполняем соответствующую операцию
  Result=Op1 Opsign Op2;
  // помещаем результат в стек
  Push(Result);
}; // конец оператора if
} // конец оператора For
```

Преобразование инфиксных выражение в постфиксные

Будем использовать:

- Стек для хранения операций и скобок
- Строку PostfixExp для формирования постфиксного выражения

Преобразование инфиксных выражение в постфиксные

Алгоритм:

- Если встретился операнд – помещаем его в строку
- Если встретилась '(' – помещаем в стек
- Если встретился оператор:
 - Если стек пуст – помещаем оператор в стек
 - Если стек не пуст – операторы более высокого приоритета выталкиваются и помещаются в строку, пока не встретится '(' или оператор более низкого приоритета
- Если встретился символ ')', то все элементы выталкиваются из стека и помещаются в строку, пока не встретится соответствующая '('
- Достигнув конца строки, все элементы стека добавляются в строку

Пример: $A-(B+C*D)/F$

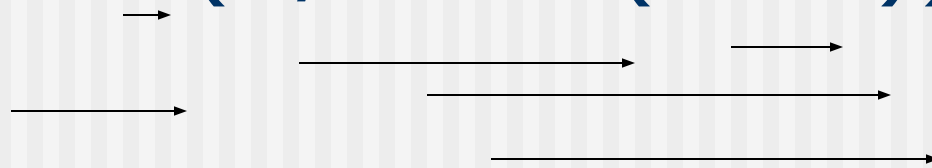
Символ	Стек	Строка
A		A
-	-	
(-(A
B	-(AB
+	-(+	AB
C	-(+	ABC
*	-(+*	ABC
D	-(+**	ABCD
)	-(+ -(-	ABCD* ABCD*+ ABCD*+
/	-/ -	ABCD*+ ABCD*+F
F	-/ -	ABCD*+F
		ABCD*+F/-

Пример: $A + B * (C / B + Z * (A + D))$

Символ	Стек	Строка	Символ	Стек	Строка
A		A	*	+*(+*	ABCB/Z
+	+	A	(+*(+*(ABCB/Z
B	+	AB	A	+*(+*(ABCB/ZA
*	+*	AB	+	+*(+*(*	ABCB/ZA
(+*(AB	D	+*(+*(+	ABCB/ZAD
C	+*(ABC)	+*(+*	ABCB/ZAD+
/	+*(/	ABC)		ABCB/ZAD+*+* +
B	+*(/	ABCB			
+	+*(+	ABCB/			
Z	+*(+	ABCB/Z			

Пример: $A+B*(C/B+Z*(A+D))$

$A+B*(C/B+Z*(A+D))$



Результат: $ABC/B/ZAD+*+*+$

Псевдокод алгоритма:

```
For (для каждого символа в стрске ch)
{ Swith (ch)
  { case operand:
    PostfixExp= PostfixExp+ch;
    break;
  case '(' :
    Push(ch);
    break;
  case ')'':
    While( '(')
    { PostfixExp= PostfixExp + getTop();
      Pop();
    };
  }
```

Псевдокод алгоритма:

```
case operator:
    While (!IsEmpty() и значение вершины != '('
           и Приоритет ch не превосходит
приоритета вершины) { PostfixExp=
PostfixExp+getTop();
    Pop();
    } // конец While
    Push(ch);
break;
} // конец Switch;
} // конец For
While(! IsEmpty() )
    { PostfixExp= PostfixExp + getTop();
      Pop();
    }; // конец While
```