

Использование шаблона «Стек»

2013

Интерфейс шаблона «Стек»

```
template <typename T>
class Stack {
void push (const T&);           //в стек
T pop ();                       //из стека
T top () const;
bool isEmpty() const ;
  bool isFull() const ;
}
```

Простейший пример

- Правильность расстановки скобок в выражении
- Открывающиеся скобки кладем на стек
- Если скобка - закрывающая и на стеке есть ей парная – удаляем парную с вершины стека. Иначе – ошибка
- Если скобки кончились, а стек не пуст – ошибка

Запись выражений

- ПОЛИЗ – бесскобочная запись

$$(2+4)*3-2*5$$

$$2\ 4\ +\ 3\ *\ 2\ 5\ *\ -$$

Алгоритм вычисления выражения в ПОЛИЗ

- Предполагается, что выражение содержит операнды - целые (однобайтные) числа без знака и знаки арифметических операций
- Будем вводить все символы без разделителей и используем символ «точка» как признак конца ввода

$24+3*25*-.$

Предполагаем, что выражение правильно
записано

Алгоритм вычисления выражения в ПОЛИЗ

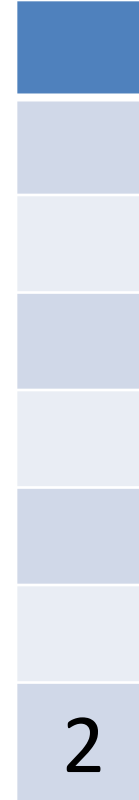
- Если символ в ПОЛИЗ – операнд (в нашем случае – цифра), то помещаем его в стек
- Если символ – знак операции – выталкиваем из стека два верхних элемента – это операнды, выполняем над ними операцию, а результат помещаем в стек
- Когда все символы выражения пройдены, на вершине стека останется результат.

Пример

2 4 + 3 * 2 5 * - .



помещаем в стек

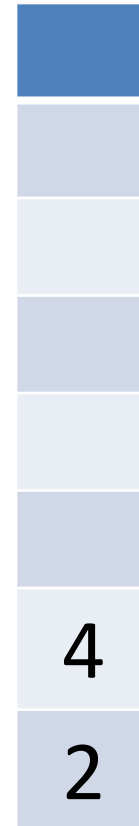


Пример

2 4 + 3 * 2 5 * - .



извлекаем
операнды из
стека



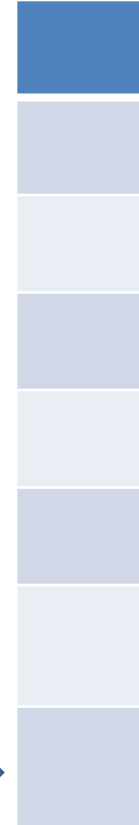
Пример

2 4 + 3 * 2 5 * - .



2 + 4

(первым идет правый операнд)



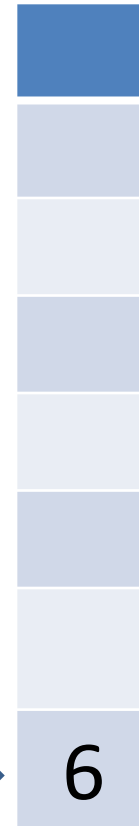
Пример

2 4 + 3 * 2 5 * - .



$$2 + 4 = 6$$

помещаем результат в стек

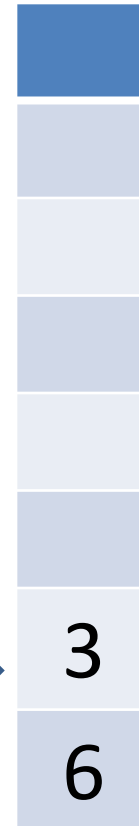


Пример

2 4 + 3 * 2 5 * - .




помещаем в стек

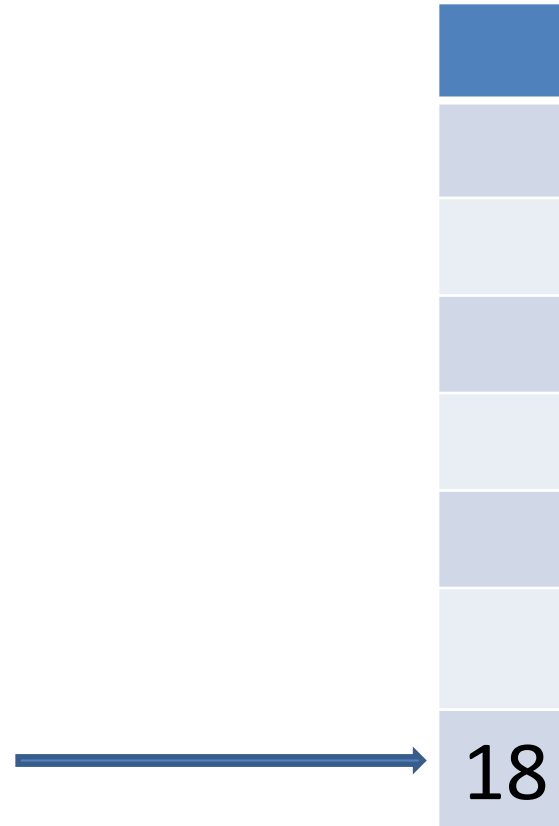


Пример

2 4 + 3 * 2 5 * - .



$6 * 3 = 18$ помещаем в стек

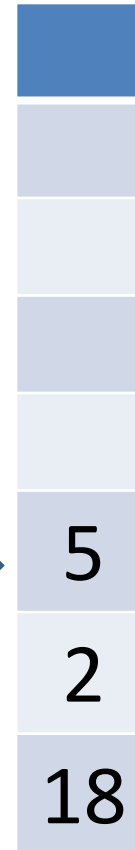


Пример

2 4 + 3 * 2 5 * - .



помещаем в стек

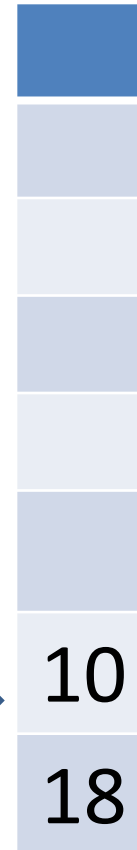


Пример

2 4 + 3 * 2 5 * - .



2* 5 помещаем в стек

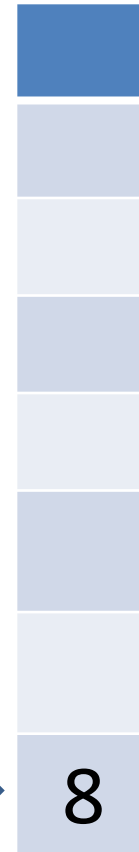


Пример

2 4 + 3 * 2 5 * _ .




18 - 10 помещаем в стек

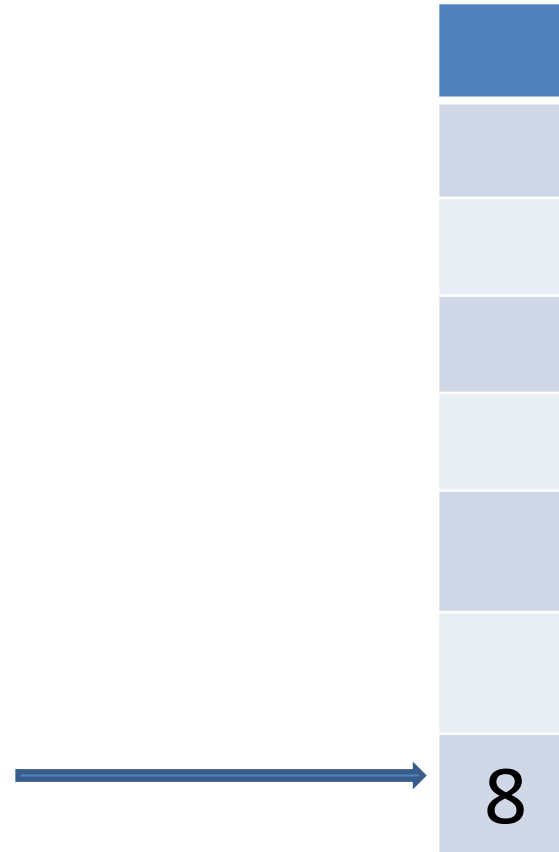


Пример

2 4 + 3 * 2 5 * — .



извлекаем результат из
стека



Еще размышления

- Будем рассчитывать на безошибочную запись выражения в ПОЛИЗ
- В стеке используем тип `double`
- Дополнительно опишем функцию вычисления результата арифметической операции

```
double execute (double a, double b, char c)
{
    switch (c) {
        case '+' : return a + b;
        case '-' : return a - b;
        case '*' : return a * b;
        case '/' : return a / b;
    }
}
```

```
# include "stack.h"
```

```
int main()
```

```
{
```

```
    Stack <double> st(25);
```

```
    char s;
```

```
    double x1,x2;
```

```
cin>>s;
while (s!= '.')
{
    if ((s>='0') && (s<='9'))    st.push (s - '0' );
    else
    { x2 = st.pop();    x1 = st.pop();
      st.push (execute (x1,x2,s));
    }
    cin>>s;
}
cout<<st.pop();
```

Как сделать, чтобы операнды были - любые

- Чтобы избежать сложного анализа строки можно считать, что операнды выражения в ПОЛИЗ - переменные (односимвольные)
- дополнительно есть таблица, где хранятся имя переменной и ее значение
- При выборе из ПОЛИЗ символа-переменной в стек помещается ее значение
- Для примера

$ab+c*de*-$, где $a=2$ $b=4$ $c=3$ $d=2$ $e=5$

Перевод выражения в ПОЛИЗ

- Выражение правильно записано
- Операнды – односимвольные переменные
- Не допускаются одноместные операции + и –
- Введем числовое определение приоритета знака операции, скобки тоже относятся к знакам операций

```
int prior ( char s){  
    switch (s) {  
        case '(' :  
        case ')' : return 0;  
        case '+' :  
        case '-' : return 1;  
        case '*' :  
        case '/' : return 2;  
    }  
}
```

```
Stack <char> st(25);
```

```
char s;
```

```
string poliz;
```

```
cin>>s;
```

```
while (s!='.')
```

```
{ // анализ: символ операнд (буква)
```

```
  // или знак операции
```

```
  //операнды помещаются в ПОЛИЗ
```

```
    if ((s>='a') && (s<='z')) poliz += s;
```



```

else { // операции и скобки
    if (s=='(') st.push(s); //всегда помещаем на стек
    else {
        while (!st.isEmpty() && prior(s) <= prior
(st.top()))
            {
                if (st.top() != '(') poliz += st.pop();
                else st.pop(); // скобка уничтожается
            }
        if (s!='(')
            st.push(s); //операция помещается в стек
    }
}

```

```
//продолжаем вводить до символа «точка»  
    cin>>s;  
}  
while (!st.isEmpty()) poliz+=st.pop();  
    //разгружаем стек в ПОЛИЗ  
  
cout << poliz <<endl;
```

Нерекурсивный обход дерева

```
typedef Node* pnode;  
struct Node {  
    int data;  
    pnode lt,rt;  
    Node (int a): data(a),lt(0),rt(0) {}  
}
```

```
class Tree {  
private:  
    pnode root;  
public:  
    Tree() {root=0;}  
    ~Tree();  
    void KLR();  
  
    . . .  
};
```

```

void Tree:: KLR() { //нерекурсивный обход
    if (root) {
        Stack <pnode> st(50);
        pnode p;
        st.push(root);
        while (!st.isEmpty()) {
            p=st.pop();
            cout<< p->data;
            if (p->rt) st.push(p->rt);
            if (p->lt) st.push(p->lt);
        }
    }
}

```