

Тема 2.2

Стеки

Уже полшестого утра...

Ты знаешь, где сейчас твой указатель стека?

Аноним.

Содержание

1. АТД «Стек»
2. Решение задачи проверки правильности расстановки скобок
3. Решение задачи вычисления арифметических выражений

1. АД «Стек»

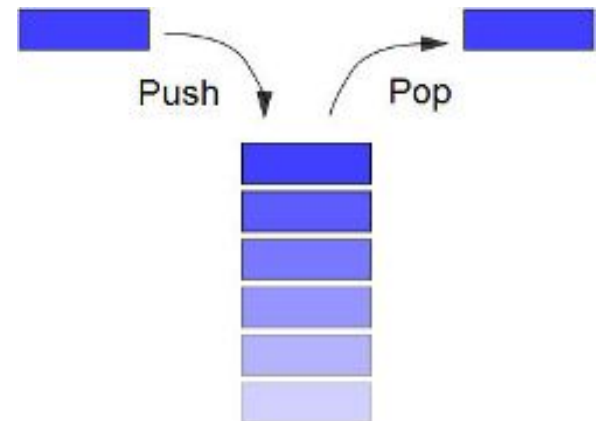
Определение

Стек – это линейная структура данных, в которой добавление и удаление элементов возможно только с одного конца (вершины стека).

Stack (Stapel) = кипа, куча, стопка (англ.)

Т.о., **стек** – это специальный тип списка, в котором все вставки (*push*) и удаления (*pop*) выполняются только на одном конце, называемом **вершиной** (*top*).

Стеки также иногда называют "магазинами", а в англоязычной литературе для обозначения стеков еще используется аббревиатура **LIFO** (last-in-first-out - последний вошел – первый вышел).



Определение

Интуитивными моделями стека могут служить

- ✓ колода карт на столе при игре в покер,
- ✓ книги, сложенные в стопку,
- ✓ или стопка тарелок на полке буфета.

Во всех этих моделях взять можно только верхний предмет, а добавить новый объект можно, только положив его на верхний.



Определение

Области применения стека:

- передача параметров в функции;
- трансляция (синтаксический и семантический анализы, генерация кодов и т.д.);
- реализация рекурсии в программировании;
- реализация управления динамической памятью и т.п.

Определение

Логическая структура стека представлена на рисунке.

- Элементы стека могут иметь как одинаковые, так и различные размеры.
- Последний добавленный в стек элемент e_n , называется *вершиной стека* (top of stack).
- Важной составляющей структуры стека является указатель, направленный к вершине, – этот указатель называется *указателем стека* (stack pointer).
- Элемент, непосредственно примыкающий к нижней границе, называется *дном стека* (bottom of stack).



Операторы, выполняемые над стеком

1. *MAKENULL(S)*.

- Делает стек **S** пустым.

2. *TOP(S)*.

- Возвращает элемент из вершины стека **S**.
- Обычно вершина стека идентифицируется позицией 1, тогда *TOP(S)* можно записать в терминах общих операторов списка как

RETRIEVE(FIRST(S), S).

3. *POP(S)*.

- Удаляет элемент из вершины стека (выталкивает из стека).
- В терминах операторов списка этот оператор можно записать как

DELETE(FIRST(S), S).

Операторы, выполняемые над стеком

4. *PUSH(x, S)*.

- Вставляет элемент **x** в вершину стека **S** (заталкивает элемент в стек).
- Элемент, ранее находившийся в вершине стека, становится элементом, следующим за вершиной, и т. д.
- В терминах общих операторов списка данный оператор можно записать как

INSERT(x, FIRST(S), S).

5. *EMPTY(S)*.

- Эта функция возвращает значение **✓true** (истина), если стек **S** пустой, **✓и** значение **false** (ложь) в противном случае.

Пример

Все текстовые редакторы имеют определенные символы, которые служат в качестве стирающих символов,

- т.е. таких, которые удаляют (стирают) символы, стоящие перед ними;
- эти символы "работают" как клавиша <Backspace> на клавиатуре компьютера.

Например, если символ **#** определен стирающим символом,

- ✓ то строка **abc#d##e** в действительности является строкой **ae**.

Текстовые редакторы имеют также символ-убийцу, который удаляет все символы текущей строки, находящиеся перед ним.

- ✓ В этом примере в качестве символа-убийцы определим символ **@**.

Пример

Операции над текстовыми строками часто выполняются с использованием стеков.

Текстовый редактор поочередно считывает символы.

- Если считанный символ не является ни символом-убийцей, ни стирающим символом, то он помещается в стек.
- Если вновь считанный символ – стирающий символ, то удаляется символ в вершине стека.
- В случае, когда считанный символ является символом-убийцей, редактор очищает весь стек.

В листинге на след. слайде представлена программа, реализующая действия стирающего символа и символа-убийцы.

Пример

```
procedure EDIT;
var   s: STACK; c: char;
begin
    MAKENULL(S);
    while not eoln do begin
        read(c);
        if c = '#' then POP(S)
        else if c = '@' then MAKENULL(S)
            else { c — ОБЫЧНЫЙ СИМВОЛ }
                PUSH(c, S)
    end;
    печать содержимого стека S в обратном порядке
end; { EDIT }
```

В этом листинге используется стандартная функция языка Pascal *eoln*, возвращающая значение **true**, если текущий символ – символ конца строки.

Пример

В этой программе тип **STACK** можно объявить как список символов.

Процесс вывода содержимого стека в обратном порядке в последней строке программы требует небольшой хитрости.

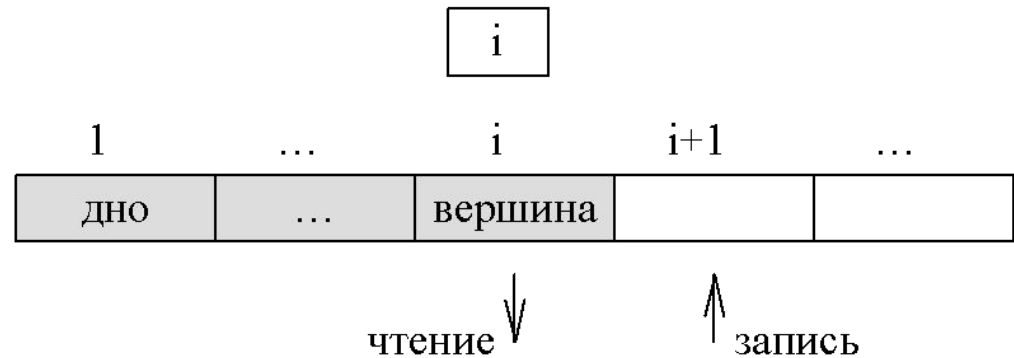
- Выталкивание элементов из стека по одному за один раз в принципе позволяет получить последовательность элементов стека в обратном порядке.
- Некоторые реализации стеков, например с помощью массивов, как описано ниже, позволяют написать простые процедуры для печати содержимого стека, начиная с обратного конца стека.
- В общем случае необходимо извлекать элементы стека по одному и вставлять их последовательно в другой стек, затем распечатать элементы из второго стека в прямом порядке.

Стеки могут представляться в памяти

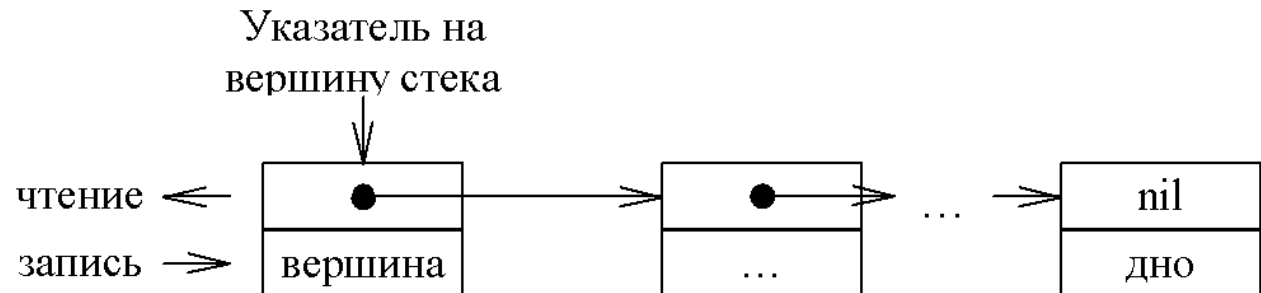
- либо в виде вектора (массива) – статическая реализация,
- либо в виде связанного списка – динамическая реализация.



Статическая реализация



Динамическая реализация



Реализация стеков

При векторном представлении под стек отводится сплошная область памяти, достаточно большая, чтобы в ней можно было поместить некоторое максимальное число элементов, которое определяется решаемой задачей.

- Граничные адреса этой области являются параметрами физической структуры стека – вектора.
- В процессе заполнения стека место последнего элемента (его адрес) помещается в указатель вершины стека.

Реализация стеков

- Если указатель выйдет за верхнюю границу стека, то стек считается переполненным и включение нового элемента становится невозможным.
- Поэтому для стека надо отводить достаточно большую память.
- Но если стек в процессе решения задачи заполняется только частично, то память используется неэффективно.

Так как под стек отводится фиксированный объем памяти, а количество элементов переменное, то говорят, что стек в векторной памяти – это ***полустатическая структура данных***.

Обычно в стеке элементы имеют один и тот же тип, поэтому обработка такого стека достаточно проста.

Реализация стеков

При списковом представлении стека память под каждый элемент стека получают динамически.

- Включение и выборка элемента осуществляются с начала списка, которое одновременно является вершиной стека.
- Переполнение стека в этом случае не происходит.
- Однако алгоритмы обработки сложнее, а время обработки удлиняется,
 - ✓ так как операции включения и выборки элементов сопряжены с обращением к операционной системе для получения или освобождения памяти.

Реализация стеков массивами

Каждую реализацию списков можно рассматривать как реализацию стеков, т.к. стеки с их операторами являются частными случаями списков с операторами, выполняемыми над списками.

- Надо просто представить стек в виде однонаправленного списка,
 - ✓ так как в этом случае операторы **PUSH** и **POP** будут работать только с ячейкой заголовка и первой ячейкой списка.
- Фактически заголовок может быть и указателем, а не полноценной ячейкой,
 - ✓ поскольку стеки не используют такого понятия, как "позиция",
 - ✓ и, следовательно, нет необходимости определять позицию 1 таким же образом, как и другие позиции.

Реализация стеков массивами

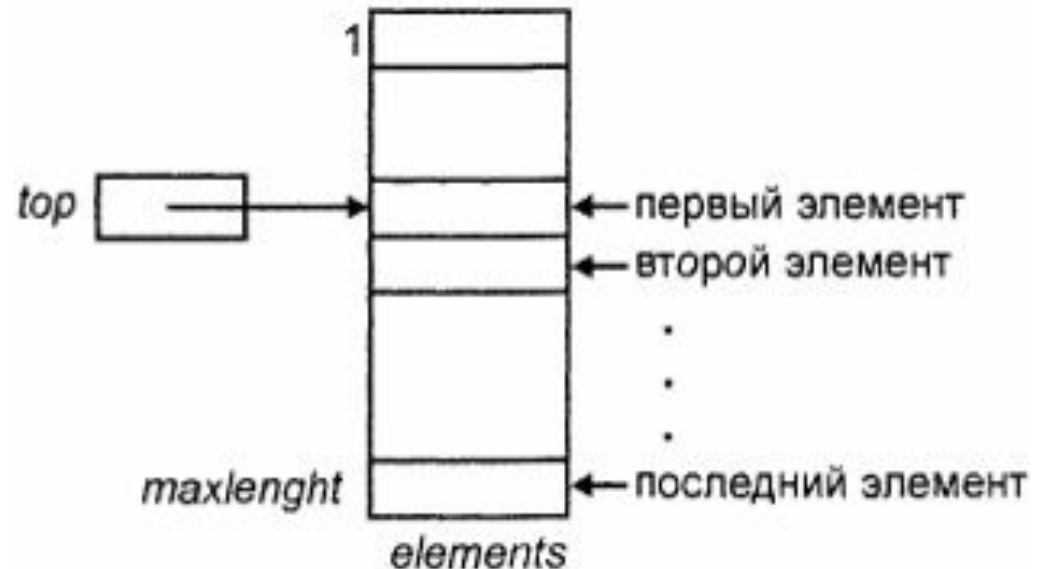
Однако реализация списков на основе массивов не очень подходит для представления стеков,

- ✓ так как каждое выполнение операторов **PUSH** и **POP** в этом случае требует перемещения всех элементов стека
- ✓ и поэтому время их выполнения пропорционально числу элементов в стеке.

Реализация стеков массивами

Можно более рационально приспособить массивы для реализации стеков, если принять во внимание тот факт, что вставка и удаление элементов стека происходит только через вершину стека:

- можно зафиксировать "дно" стека в самом низу массива (в ячейке с наибольшим индексом)
- и позволить стеку расти вверх массива (к ячейке с наименьшим индексом);
- переменная с именем **top** (вершина) будет указывать положение текущей позиции первого (верхнего) элемента стека.



Реализация стеков массивами

Для такой реализации стеков можно определить абстрактный тип *STACK* следующим образом:

```
type  
  STACK = record  
    top: integer;  
    element: array[1..maxlength] of elementtype  
  end;
```

В этой реализации стек состоит из последовательности элементов

element[*top*], *element*[*top* + 1], ...,
element[*maxlength*].

Если *top* = *maxlength*+1, то стек пустой.

Реализация стеков массивами: операторы

```
procedure   MAKENULL ( var S: STACK );  
begin  
    S.top := maxlength + 1  
end;        { MAKENULL }  
  
function EMPTY ( S: STACK ): boolean;  
begin  
    if    S.top > maxlength  then  
        EMPTY := true  
    else   EMPTY := false  
end;      { EMPTY }
```

Реализация стеков массивами: операторы

```
function TOP ( var S: STACK ): elementtype;  
    { Возвращает элемент из вершины стека S. }
```

```
begin
```

```
    if EMPTY(S) then
```

```
        error('Стек пустой')
```

```
    else TOP := S.elements[S.top]
```

```
end;    {TOP}
```

```
procedure POP ( var S: STACK );
```

```
    { Удаляет элемент из вершины стека (выталкивает из стека). }
```

```
begin
```

```
    if EMPTY(S) then
```

```
        error('Стек пустой')
```

```
    else S.top := S.top + 1
```

```
end;    {POP}
```

Реализация стеков массивами: операторы

```
procedure PUSH ( x: elementtype; var S: STACK );  
{ Вставляет элемент x в вершину стека S (заталкивает элемент в стек). }  
begin  
    if    S.top = 1 then    error('Стек полон')  
    else    begin  
        S.top := S.top - 1  
        S.elements[S.top] := x  
    end  
end;          { PUSH }
```

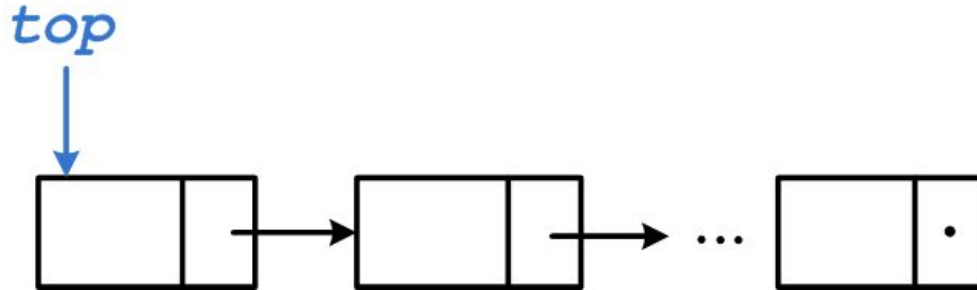

Реализация стеков списком

Стек как динамическую структуру данных легко организовать на основе линейного списка.

- Поскольку работа всегда идет с заголовком стека, то есть не требуется осуществлять
 - ✓ просмотр элементов,
 - ✓ удаление
 - ✓ и вставку элементов в середину или конец списка, то достаточно использовать экономичный по памяти линейный однонаправленный список.
- Для такого списка достаточно хранить указатель вершины стека, который указывает на первый элемент списка.
- Если стек пуст, то списка не существует и указатель принимает значение *nil*.

Реализация стеков списком

Логическая схема стека на основе линейного списка:



```
type celltype = record
  element: elementtype;
  next: ^ celltype
end;
STACK = ^ celltype;

var top: STACK;
```

Реализация стеков списком

Реализация основных операций со стеком приведена в виде соответствующих процедур.

Приведены их два варианта:

- с использованием процедур операций с линейным однонаправленным списком;
- самостоятельная реализация – без таких процедур.

Реализация стеков списком

Реализация стека на базе линейного однонаправленного списка:

```
procedure Push(x: elementtype;  
                var top: STACK);  
    {Запись элемента в стек (положить в стек)}  
begin  
    InsFirst_SingleList(x, top);  
end;    { PUSH }  
  
procedure Pop(var x: elementtype;  
                var top: STACK);  
    {Чтение элемента из стека (снять со стека) с удалением }  
begin  
    if top <> nil then begin  
        x := top^.element;  
        Del_SingleList(top, top);    {удаляем вершину}  
    end;  
end;    { POP }
```

Реализация стеков списком

```
procedure MAKENULL(var top: STACK);  
    {Очистка стека}  
begin  
    while top <> nil do  
        Del_SingleList(top, top);    {удаляем вершину}  
end;    { MAKENULL }  
  
function Empty(top: STACK): boolean;  
    {Проверка пустоты стека}  
begin  
    if top = nil then Empty := true  
        else Empty := false;  
end;    { EMPTY }
```

Реализация стеков списком

Реализация стека отдельными процедурами:

```
function Empty( top: STACK) :boolean;
begin
    Empty:= Top=nil;
end;      { EMPTY }

procedure Push(x: elementtype; var top: STACK);
    {добавление элемента в стек}
Var p: STACK;
begin
    new(p); {создаем новый узел}
    p^.next:= top; {он будет находиться перед вершиной}
    p^.element:= x;
    top:=p; {Делаем p вершиной стека}
end;      { PUSH }
```

Реализация стеков списком

```
procedure Pop(var top: STACK; var x: elementtype);
  {Возвращает число из вершины стека и затем уничтожает вершину}
Var p: STACK;
begin
  if Empty(top)=False then begin
    p:= top^.next;           {запоминаем следующий узел}
    x:=top^.element;       {вытаскиваем инф-цию из вершины}
    dispose(top);           {уничтожаем вершину}
    top:=p;                 {делаем p вершиной}
  end else
    error('стек уже пуст')
end;           { POP }
```

2. Решение задачи проверки правильности расстановки скобок

Задача

Задача: вводится символьная строка, в которой записано выражение со скобками трех типов: [], {} и (). Определить, верно ли расставлены скобки (не обращая внимания на остальные символы). Примеры:

[(())}] [[({})]

Упрощенная задача: то же самое, но с одним видом скобок.

Решение: счетчик вложенности скобок. Последовательность правильная, если в конце счетчик равен нулю и при проходе не разу не становился отрицательным.

(()) ()

1 2 1 0 1 0

(())) (

1 2 1 0 -1 0

(()) (

1 2 1 0 1

[({)] }

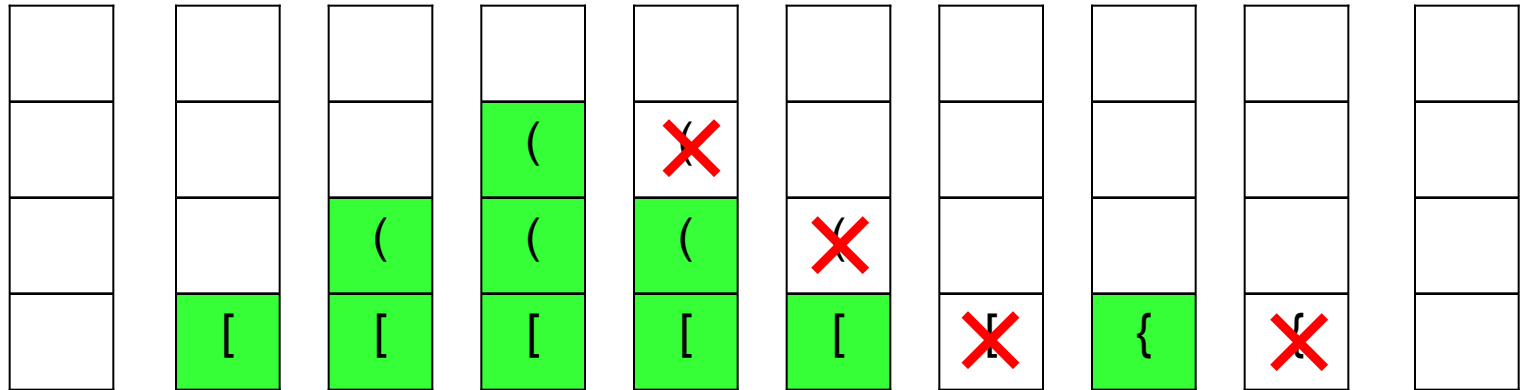
(: 0	1	0	
[: 0	1		0
{: 0		1	0



Можно ли решить исходную задачу так же, но с тремя счетчиками?

Решение задачи со скобками

[(())] { }



Алгоритм:

- 1) в начале стек пуст;
- 2) в цикле просматриваем все символы строки по порядку;
- 3) если очередной символ – открывающая скобка, заносим ее на вершину стека;
- 4) если символ – закрывающая скобка, проверяем вершину стека: там должна быть соответствующая открывающая скобка (если это не так, то ошибка);
- 5) если в конце стек не пуст, выражение неправильное.

Реализация стека (массив)

Структура-стек:

```
const MAXSIZE = 100;  
type Stack = record { стек на 100 символов }  
  data: array[1..MAXSIZE] of char;  
  size: integer; { число элементов }  
end;
```

Добавление элемента:

```
procedure Push( var S: Stack; x: char);  
begin  
  if S.size = MAXSIZE then  
    writeln('Стек полон')  
  else begin  
    S.size := S.size + 1;  
    S.data[S.size] := x;  
  end;  
end;
```

ошибка:
переполнение
стека

добавить элемент

Реализация стека (массив)

Снятие элемента с вершины:

```
function Pop ( var S:Stack ): char;  
begin  
  if S.size = 0 then begin  
    Pop := char(255);  
  end else begin  
    Pop := S.data[S.size];  
    S.size := S.size - 1;  
  end;  
end;
```

ошибка:
стек пуст

Пустой или нет?

```
Function Empty ( S: Stack ): Boolean;  
begin  
  Empty := (S.size = 0);  
end;
```

Программа

```
var br1, br2, expr: string;
    i, k: integer;
    upper: char;      { то, что сняли со стека }
    error: Boolean;  { признак ошибки }
    S: Stack;
begin
    br1 := '([';    br2 := ')]';
    S.size := 0;
    error := False;
    writeln('Введите выражение со скобками');
    readln(expr);
    ... { здесь будет основной цикл обработки }
    if not error and Empty(S) then
        writeln('Выражение правильное.')
    else writeln('Выражение неправильное.')
end.
```

открывающие скобки

закрывающие скобки

Обработка строки (основно

ЦИКЛ ПО ВСЕМ СИМВОЛАМ строки expr

```

for i:=1 to length(expr) do begin
  for k:=1 to 3 do begin
    if expr[i] = br1[k] then begin { откр. скобка }
      Push(S, expr[i]); { втолкнуть в стек }
      break;
    end;
    if expr[i] = br2[k] then begin { закр. скобка }
      upper := Pop(S); { снять символ со стека }
      error := upper <> br1[k];
      break;
    end;
  end;
  if error then break;
end;

```

ЦИКЛ ПО ВСЕМ ВИДАМ скобок

ошибка: стек пуст или не та скобка

была ошибка: дальше нет смысла проверять

Реализация стека (список)

Структура узла:

```
type PNode = ^Node; { указатель на узел }  
    Node = record  
        data: char; { данные }  
        next: PNode; { указатель на след. элемент }  
    end;
```

Добавление элемента:

```
procedure Push( var top: PNode; x: char);  
var NewNode: PNode;  
begin  
    New(NewNode); { выделить память }  
    NewNode^.data := x; { записать символ }  
    NewNode^.next := top; { сделать первым узлом }  
    top := NewNode;  
end;
```

Реализация стека (список)

Снятие элемента с вершины:

```
function Pop ( var top: PNode ): char;  
var q: PNode;  
begin  
  if top = nil then begin { стек пуст }  
    Pop := char(255); { неиспользуемый символ }  
  end else begin  
    Pop := top^.data; { взять верхний символ }  
    q := top; { запомнить вершину }  
    top := top^.next; { удалить вершину из стека }  
    Dispose (q); { удалить из памяти }  
  end;  
end;
```



Можно ли переставлять операторы?

Реализация стека (список)

Пустой или нет?

```
function Empty ( S: Stack ): Boolean;
begin
  Empty := (S = nil);
end;
```

Изменения в основной программе:

~~*var S: Stack;*~~

var S: PNode;

~~*...*~~

~~*S.size = 0;*~~

S := nil;



Больше ничего не меняется!

3. Решение задачи вычисления арифметических выражений

Одно из применений стеков можно продемонстрировать на примере вычисления значения арифметического выражения в калькуляторах.

Пусть арифметическое выражение составлено из комбинации

- чисел,
- знаков бинарных арифметических операций (операторов)
✓ +, -, *, /, ^,
- круглых скобок (,)
- и пробелов.

Алгоритм вычисления предусматривает представление выражения в определенном формате.

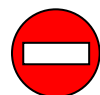
Формы записи арифметических выражений

Как вычислять автоматически?

$$(a + b) / (c + d - 1)$$

Инфиксная запись

(знак операции между



необходимы скобки! операндами)

Префиксная запись (знак операции до операндов)

$$/ + a b - + c d 1$$

польская нотация,
[Jan Łukasiewicz](#) (1920)



скобки не нужны, можно однозначно
вычислить!

Постфиксная запись (знак операции после операндов)

$$a b + c d + 1 - /$$

обратная польская нотация,
[F. L. Bauer](#) F. L. Bauer and [E. W.](#)

[Dijkstra](#)

Формы записи арифметических выражений⁴⁵

В калькуляторах чаще всего применяется представление выражения в

- инфиксной форме – для записи исходного выражения;
- и постфиксной форме – для автоматического вычисления выражения.

В обеих формах выражения представляются в виде символьных строк.

Формы записи арифметических выражений⁴⁶

В **инфиксной форме** записи каждый бинарный оператор помещается между двумя своими операндами.

- Для уточнения порядка вычислений могут использоваться круглые скобки.
- Инфиксный формат записи используется в большинстве языков программирования и калькуляторах и практически совпадает с естественной формой записи выражения.

Примеры записи выражений:

✓ $5.7 + 6.8 =$

✓ $15 * 4 + (25 / 2 - 3) ^ 2 =$

✓ $3 * 7.5 + 6e2 / 5.0 =$

Формы записи арифметических выражений⁴⁷

В **постфиксной форме** записи (обратной польской записи ОПЗ, или Reverse Polish Notation RPN) операнды предшествуют своему оператору.

Примеры записи выражений:

✓ $5.7 \ 6.8 \ + \ =$

✓ $15 \ 4 \ * \ 25 \ 2 \ / \ 3 \ - \ ^ \ + \ =$

✓ $3 \ 7.5 \ * \ 6e2 \ 5.0 \ / \ + \ =$

Постфиксный калькулятор

Наиболее простым является алгоритм вычисления постфиксного выражения.

- Исходная строка содержит элементы только двух видов: числа и операторы.
- Пусть выражение заканчивается символом '='.
- Алгоритм использует один стек, элементами которого являются числа вещественного типа.

Постфиксный калькулятор

Алгоритм вычисления постфиксного калькулятора.

- ❶ Из исходной строки выделяется очередной элемент.
- ❷ Если элемент – число, то оно заносится в стек.
Переход к п.1.
- ❸ Если элемент – оператор, то из стека последовательно извлекаются два элемента,
 - сначала правый операнд,
 - затем левый операнд
 - и над ними выполняется операция, определенная оператором.
 - Результат операции (число) заносится в стек.
 - Переход к п.1.
- ❹ Пункты 1 – 3 выполняются до тех пор, пока в исходной строке не встретится признак конца выражения '='.

Постфиксный калькулятор

Рассмотрим порядок вычисления выражения

$$3 \ 7.5 \ * \ 6e2 \ 5 \ / \ + \ =$$

- Шаг 1.** Из строки выделяется число 3 и помещается в стек. Стек: 3.
- Шаг 2.** Из строки выделяется число 7.5 и помещается в стек. Стек: 3 7.5.
- Шаг 3.** Из строки выделяется оператор '*'. Из стека извлекаются числа 7.5 и 3. Выполняется операция $3 * 7.5$, результат 22.5 помещается в стек. Стек: 22.5.
- Шаг 4.** Из строки выделяется число 6e2 и помещается в стек. Стек: 22.5 600.
- Шаг 5.** Из строки выделяется число 5 и помещается в стек. Стек: 22.5 600 5.
- Шаг 6.** Из строки выделяется оператор '/'. Из стека извлекаются два числа 5 и 600. Выполняется операция $600 / 5$, и результат 120 помещается в стек. Стек: 22.5 120.
- Шаг 7.** Из строки выделяется оператор '+'. Из стека извлекаются два числа 120 и 22.5. Выполняется операция $22.5 + 120$. Результат 142.5 засылается в стек.
- Шаг 8.** Из строки выделяется символ '=' - признак конца выражения. Из стека извлекается результат вычисления - число 142.5.

Постфиксный калькулятор

Постфиксная форма:

$X = a b + c d + 1 - /$

					d		1		
		b		c	c	c+d	c+d	c+d-1	
	a	a	a+b	a+b	a+b	a+b	a+b	a+b	X

Преобразование выражения из инфиксной формы в постфиксную

52

Алгоритм преобразования выражения из инфиксной формы в постфиксную основан на методе стека с приоритетами.

- В нем всем операторам и скобкам-разделителям ставятся в соответствие целочисленные приоритеты.
- Чем старше операция, тем выше ее приоритет.
- Открывающая скобка имеет низший приоритет, равный 0,
- закрывающая скобка – равный 1.

В ходе обработки исходной строки

- операнды переносятся в выходную строку непосредственно,
- а операторы – через стек в соответствии со своими приоритетами.

Преобразование выражения из инфиксной формы в постфиксную

53

Элемент стека состоит из двух полей:

- оператор или скобка – символьный тип;
- приоритет – целочисленный тип.

Приоритет пустого стека полагаем равным нулю.

Преобразование выражения из инфиксной формы в постфиксную

Алгоритм метода.

- ❶ Из исходной строки выделяется очередной элемент S_i .
- ❷ Если S_i – операнд, то записать его в выходную строку и перейти к п.1;
 - иначе перейти к п.3.
- ❸ Если приоритет S_i равен нулю
 - ✓ (т.е. элемент – открывающая скобка)
 - или больше приоритета элемента S_j , находящегося в вершине стека, то добавить S_i в вершину стека и перейти к п.4;
 - иначе перейти к п.5.

Преобразование выражения из инфиксной формы в постфиксную

- ④ Если теперь элемент в вершине стека имеет приоритет, равный 1
 - ✓ (т.е. добавленный элемент S_i является закрывающей скобкой),
то из стека удалить два верхних элемента (закрывающую и открывающую скобки);
 - перейти к п.1.
- ⑤ Элемент (оператор) из вершины стека вытолкнуть в выходную строку и перейти к п.3.
- ⑥ Пункты 1 – 5 выполнять до тех пор, пока не встретится признак конца выражения – символ '='.
 - Тогда все элементы из стека вытолкнуть в выходную строку,
 - затем занести туда символ “=”.

Преобразование выражения из инфиксной формы в постфиксную

56

Рассмотрим порядок преобразования выражения

$$15 * 4 + (25 / 2 - 3) ^ 2 = .$$

Шаг 1. Выделен операнд 15, заносим его в выходную строку.

Выходная строка: 15, стек пуст.

Шаг 2. Выделен оператор '*', его приоритет больше приоритета пустого стека, помещаем в стек.

Стек: *, выходная строка: 15.

Шаг 3. Выделен операнд 4, его в выходную строку.

Выходная строка: 15 4, стек: *.

Шаг 4. Выделен оператор '+', его приоритет меньше приоритета '*' в вершине стека, поэтому '*' выталкиваем в выходную строку, а '+' заносим в стек.

Выходная строка: 15 4*, стек: +.

Шаг 5. Выделена открывающая скобка с нулевым приоритетом, помещаем его в стек.

Выходная строка: 15 4*, стек: + (.

Шаг 6. Выделен операнд 25, помещаем в выходную строку.

Выходная строка: 15 4 * 25, стек: + (.

Преобразование выражения из инфиксной формы в постфиксную 57

$$15 * 4 + (25 / 2 - 3) ^ 2 = .$$

Шаг 7. Выделен оператор “/”, его приоритет выше приоритета ‘(’. помещаем в стек.

Выходная строка: 15 4 * 25, стек: + (/.

Шаг 8. Выделен операнд 2, его в строку: 15 4 * 25 2.

Шаг 9. Выделен оператор ‘-’, его приоритет меньше приоритета “/” из вершины стека, поэтому “/” - в строку, теперь приоритет ‘-’ больше приоритета ‘(’ и ‘-’ заносим в стек.

Выходная строка 15 4 * 25 2 /, стек: + (-.

Шаг 10. Выделен операнд 3, его в строку: 15 4 * 25 2 / 3.

Шаг 11. Выделена закрывающая скобка ‘)’, ее приоритет меньше приоритета ‘-’ из стека, поэтому ‘-’ - в строку.

Теперь приоритет ‘)’ больше приоритета ‘(’, помещаем ‘)’ в стек, но так как его приоритет равен 1, то из стека удаляем два элемента: ‘)’ и ‘(’ без занесения их в выходную строку).

Выходная строка: 15 4 * 25 2 / 3 -, стек: +.

Шаг 12. Выделен оператор ‘^’, его приоритет больше приоритета ‘+’, ‘^’ засылаем в стек: + ^, строка без изменения.

Преобразование выражения из инфиксной формы в постфиксную 58

$$15 * 4 + (25 / 2 - 3) ^ 2 = .$$

Шаг 13. Выделен операнд 2, его - в строку. 15 4 * 25 2 / 3 - 2.

Шаг 14. Выделен признак конца выражения '=', из стека выталкиваем в строку '^' и '+', затем в строку заносим '='.

Выходная строка сформирована полностью: 15 4 * 25 2 / 3 - 2 ^ + =,
стек пуст.

Инфиксный калькулятор

Очевидно, что, используя рассмотренные выше алгоритмы

- ✓ преобразования инфиксного выражения в постфиксное
 - ✓ и вычисления постфиксного выражения,
- легко создать инфиксный калькулятор.

Его алгоритм будет основан на применении двух стеков:

- стека операторов в алгоритме преобразования
- и стека операндов в алгоритме вычисления.

Сам алгоритм будет состоять из двух самостоятельных частей, выполняемых последовательно.

Первая часть преобразует инфиксную строку в постфиксную, которая является входом для второй части, выполняющей вычисление постфиксного выражения.

Инфиксный калькулятор

Более подходящим является алгоритм, в котором рассмотренные выше алгоритмы выполняются не последовательно, а совместно:

- по мере того как часть инфиксной строки преобразуется в постфиксную, осуществляется вычисление преобразованной части выражения.

Такой подход

- ✓ облегчает проверку правильности исходного выражения
- ✓ и позволяет прекратить его обработку при выявлении ошибок.