Реализация стека на массивах

Стек — важная структура данных в программировании. Он предназначен для временного хранения данных, которые извлекаются из стека в порядке, обратном их добавлению.
Стек широко используется в программировании:

- для хранения адресов возврата в программу из подпрограмм или из процедур обработки прерываний
- для передачи параметров в процедуры (функции)
- для размещения локальных параметров процедур и функций
- для реализации работы алгоритмов, имеющих рекурсивный характер.

Язык Pascal не имеет типа данных стек, поэтому для его моделирования наиболее подходящими структурами являются:

- массивы (**статический стек**)
- указатели (**динамический стек).**

Операции с массивами выполняются значительно быстрее, нежели переход по указателям, поэтому в этой статье рассматривается реализация стека на массивах. Используемый язык программирования – Pascal.

Определение стека

Стек (stack (англ.) - стопка**)** – *упорядоченный набор элементов, в котором добавление новых элементов и удаление существующих производится с одного конца, называемого* ***вершиной стека (top)*****.**

Стек функционирует по принципу **LIFO** (Last - In - First- Out) - "**последним пришел - первым исключается**".
Понятие стека в 1946 ввел Алан Тьюринг.

Условно стек можно представить в виде стакана с шариками.

При записи и выборке изменяется только адрес вершины стека. Поэтому каждый стек имеет базовый адрес (**top**), от которого производятся все операции со стеком. Эта переменная будет определять индекс элемента , который в данный момент времени находится в вершине стека. В случае, когда стек пуст (**top=0**), адреса вершины и основания стека совпадают.

Операции со стеком

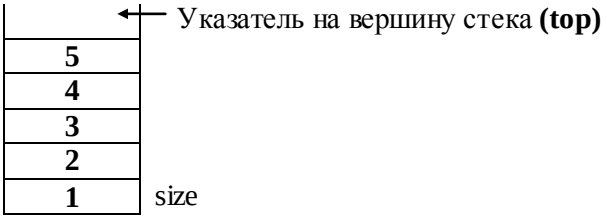
Основные операции со стеком:

- запись элемента в стек (**push** – «вталкивание»);
- извлечение элемента из стека (**pop** – «выталкивание»);
- проверка наличия элементов в стеке (**empty** - «стек пустой» или **full** - «переполнение стека»).

Рассмотрим, как реализовать стек на базе массива. Будем считать, что для реализации стека используется массив статического типа. Тип компонент и размерность массива зависят от конкретной задачи.
Некоторые трудности при этом возникают в силу того, что число элементов стека не известно, а число элементов массива ограничено. Поэтому при объявлении размерности массива надо указать предполагаемый максимальный размер стека. Например, если мы уверены, что число элементов в стеке не превысит 100, то надо использовать массив из 100 элементов.

```
const size=100;
var a:array[0..size]of integer;
    i,x,n,top:integer;
```

Рассмотрим реализацию стека, когда вершина стека находится наверху:



Пусть вводятся последовательно элементы 1, 2, 3, 4, 5, начиная с конца массива. Тогда стек будет заполнен следующим образом (см. рисунок). Доступный элемент для извлечения из стека будет 5. Начальное значение указателя на вершину стека равно **top+1.**

Операции «вытолкнуть» **pop** и «втолкнуть» **push** будут выглядеть следующим образом:

```
procedure pop( var x:integer);
begin top:=top+1;
      x:=a[top];
end;

procedure push(x:integer);
begin a[top]:=x;
      top:=top-1;
end;
```

Начальное состояние элементов массива должно быть равно нулю. Для этого можно использовать процедуру инициализации **init:**

```
procedure init;
var i:integer;
begin for i:=0 to size do a[i]:=0;
      top:=size;
end;
```

Для проверки пустоты **(empty)** или переполнения (**full)** стека достаточно написать следующие функции:

```
function empty:boolean;
begin if top=size then empty:=true else empty:=false;end;

function full:boolean;
begin if top=0 then full:=true else full:=false;end;
```

Для того, чтобы узнать размер заполненного стека, можно воспользоваться функцией **sizestack:**

```
function sizestack:integer;
begin sizestack:=size-top;  end;
```

Теоретически возможна и другая реализация стека через массив, когда вершина стека зафиксирована внизу. Как тогда изменятся процедуры **pop** и **push**?

Пример программы, которая иллюстрирует работу со стеком при добавлении элементов в стек и извлечении их из стека:

```
uses crt;
const size=10;
var a:array[0..size]of integer;
    i,x,n,top:integer;
procedure init;
var i:integer;
begin for i:=0 to size do a[i]:=0;
      top:=size;
end;
function full:boolean;
begin if top=0 then full:=true else full:=false;end;
function empty:boolean;
begin if top=size then empty:=true else empty:=false;end;
function sizestack:integer;
begin sizestack:=size-top;  end;
procedure push(x:integer);
begin a[top]:=x;
      top:=top-1;
end;
procedure pop( var x:integer);
begin top:=top+1;
      x:=a[top];
end;
begin writeln('input n');
      readln(n);
      init;
      i:=1;
      {Заполнение стека}
      while (i<=n) and not full do
       begin readln(x);
             push(x);
             i:=i+1;
       end;
      if full then writeln('stack full');
      {Вывод содержимого стека}
      for i:=top+1 to top+sizestack do write(a[i]:3);
      writeln;
      writeln('input n');
      readln(n);
      i:=1;
      {Извлечение элементов из стека}
      while (i<=n) and not empty do
       begin pop(x);
             write(x:3);
             i:=i+1;
       end;
      writeln;
      if not empty then for i:=top+1 to top+sizestack do write(a[i]:3)
                   else writeln ('stack empty');
end.
```

Модуль для работы со стеком

Все процедуры и функции, описанные выше, можно объединить в один пользовательский модуль (**Unit**), который можно подключить к любой программе, работающей со стеком. Это позволит сэкономить время и память при написании программ.

Пусть модуль называется **STACK (`unit stack;`)**. Подключить его к программе можно следующим образом: **`Uses stack;`**
Сам модуль будет выглядеть следующим образом:

```
unit stack;
interface
const size=10000;
type ta=array[0..size]of char;
var top:integer;
    a:ta;
procedure init;
function full:boolean;
function empty:boolean;
function sizestack:integer;
procedure push(x:char);
procedure pop(var x:char);
implementation
procedure init;
var i:integer;
begin for i:=0 to size do a[i]:=' ';
      top:=size;
end;
function full:boolean;
begin if top=0 then full:=true else full:=false;end;
function empty:boolean;
begin if top=size then empty:=true else empty:=false;end;
function sizestack:integer;
begin sizestack:=size-top;  end;
procedure push(x:char);
begin a[top]:=x;
      top:=top-1;
    end;
procedure pop(var x:char);
begin top:=top+1;
      x:=a[top];
end;
end.
```

Пример решения задачи с помощью стека

Существует множество задач, которые очень трудно решить без использования стеков и которые легко решаются с использованием этой структуры данных. Рассмотрим одну из них.

Пусть, в математическом выражении встречаются скобки трех типов:

- круглые (“(“,”)”),
- квадратные (“[“,”]”)
- фигурные (“{“,”}”).

Надо проверить, выполняется ли баланс скобок, учитывая, что закрывающая скобка должна быть того же типа, что и соответствующая ей открывающая и количество открывающих и закрывающих скобок должно совпадать.

Для решения задачи очень удобным оказывается использование стека.
Будем действовать следующим образом:

- если обнаружена открывающая скобка, то она записывается в стек;
- если обнаружена закрывающая скобка, анализируем содержимое стека:
  если стек пуст, то открывающая скобка отсутствует и выражение составлено неправильно;

- если стек не пуст, то выбираем из стека открывающую скобку и проверяем соответствует ли ее тип типу закрывающей скобки;
- если тип скобок совпадает, то продолжаем проверку, в противном случае выражение составлено неправильно.

После того, как будет просмотрена вся строка, необходимо проверить пуст ли стек. Если стек не пуст, то количество закрывающих и открывающих скобок не совпадает и выражение составлено неверно.

К программе подключен модуль stack. Вот текст программы :

```
uses stack;
var s:string;
    i,j:integer;
    x:char;
    b:boolean;
begin readln(s);
      init;
      b:=true;
      i:=1;
      while (i<=length(s)) and b do
       begin if (s[i]='{')or(s[i]='[')or(s[i]='(') then push(s[i]);
             if (s[i]='}')or(s[i]=']')or(s[i]=')') then
             if empty then b:=false else
              begin pop(x);
                    if (x='(')and(s[i]<>')') then b:=false;
                    if (x='[')and(s[i]<>']') then b:=false;
                    if (x='{')and(s[i]<>'}') then b:=false;
              end;
              i:=i+1;
       end;
      if not empty then b:=false;
      if b then writeln('Скобки расставлены верно') else writeln('Скобки
расставлены неверно');
end.
```

Используемая литература:

1. Окулов С. М. Абстрактные типы данных. Бином. 2009.
2. Поляков К.Ю., Еремин Е.А. Информатика. Углубленный уровень. Учебник для 11 класса., часть 2. Бином. 2013.

Ссылки:

1. https://ru.wikipedia.org
2. http://studopedia.net/10_99120_ispolzovanie-steka-v-programmirovanii.html
3. http://algolist.manual.ru/ds/basic/stack_recursion.php
4. http://pascal40.narod.ru/pgs/Stacs.htm

Батракова Людмила Васильевна