#### *Лабораторная работа 5.* Способы физического представления стека и очереди.

СТЕК — динамическая структура данных, организованная по принципу «Последним пришёл — первым ушёл». Другое название стека —структура LIFO ( это название происходит от англоязычного аналога указанного принципа: «Last In — First Out»). Иногда может применяться раннее название МАГАЗИН, происхождение которого связано с аналогичной организацией оружейного магазина.

Особенностями СТЕКА являются:

1. последовательный доступ к элементам, организованным в СТЕК;
2. неразрушающая запись ( при добавлении новых данных в структуру);
3. единственная точка доступа к данным, образующим СТЕК.

Последний элемент, введённый в СТЕК и ещё не удалённый, называется *вершиной* СТЕКА, остальные элементы образуют *тело* СТЕКА. Тело , в свою очередь, является СТЕКОМ.

Допустимыми операциями при работе со СТЕКОМ в классическом варианте считаются следующие:

1. создание СТЕКА:
2. проверка, является ли СТЕК пустым;
3. добавление нового элемента в СТЕК;
4. удаление элемента, находящегося в точке доступа;
5. просмотр данного, находящегося в точке доступа.

При практической реализации стека к этим операциям обычно добавляют возможность просмотра содержимого всей структуры данных и возможность очистки памяти, выделенной под стек, когда потребность в использовании стека отпала.

Стек используется, например, при обработке последовательности активных подпрограмм или вложенных циклов. В этом случае подпрограмма, вызванная последней, закончит свою работу первой. Вложенный цикл завершится ранее охватывающего цикла. Стек образуют омонимические переменные в программе. Стек применяется при работе рекурсивных функций, а также при обработке прерываний.

1.1. Цепное представление стека

Под цепным представлением понимают представление в динамически распределяемой области памяти (в «куче») в виде цепочки элементов, связанных с помощью указателей. В качестве точки входа в цепочку применяют указатель на вершину. Описание типа соответствуюшего указателя на Турбо Паскале может быть дано в форме:

Type UKAZ=^STK;

STK=record

info: integer;

next: UKAZ {указатель на предыдущий элемент стека – следующий элемент цепочки}

end;

Var Top, NE, Kon: UKAZ; value: integer;

Здесь указатель включён в структуру данных. Поле записи info имеет тип integer. Это означает, что в рассматриваемом примере тип данных, образующих СТЕК, предполагается целочисленым. В случае другого типа данных поле info следует изменить.

Далее рассматривается вариант реализации операций со стеком при цепном представлении. При этом использованы процедуры и функции( реализация на языке Турбо Паскаль).

*1. Создание стека (Top- указатель на вершину)*

Procedure Sozd\_St;

BEGIN

Top:=Nil;

END;

*Замечание.* На практике часто операцию создания стека совмещают с внесением в него начальных данных.

*2. Добавление элемента в стек (Добавляемый элемент – value)*

Procedure Dobav\_St;

BEGIN

New(NE);

NE^.next:=Top;

NE^.info:= value;

Top:= NE

END;

*3. Удаление элемента из стека*

Procedure Udal\_St;

BEGIN

Kon:= Top; {Для сохранения доступа к фрагменту памяти, занимаемому удаляемым элементом. }

Top:=Top^.next;

Dispose(Kon) {Возврат памяти в «кучу»}

END;

*4.Чтение вершины (последний элемент в стеке)*

Function Last (S:UKAZ; Var ERR: boolean): integer;

BEGIN

If S=Nil then begin

Writeln(‘стек пуст’); err:=true;

Exit

end

else begin {чтение успешно}

write(‘вершина=’, S^.info:4);

err:=false;

Last:= S^.info

end

END;

*5. Просмотр элементов стека*

Procedure Cont\_St;

{вывод элементов в обратном порядке}

BEGIN

Kon:= Top;

while Kon<>Nil do

begin

writeln(Kon^.info);

Kon :=Kon^.next

end

END;

1.2. Сплошное представление стека

При сплошном представлении элементы расположены в памяти последовательно один за другим. Обычно такое представление предполагает отображение стека на одномерный массив с базовым типом, соответствующим типу данных, размещаемых в стеке. Кроме описания этого массива требуется указатель на вершину стека. Роль такого указателя играет переменная, имеющая тип индекса массива размещения. Поскольку динамическая структура данных (стек) моделируется в статической (одномерный массив), то следует предусматривать проверку переполнения массива при добавлении данных.

Пусть стек размещается в массиве СТЕК [1..n]:T, имеющем n элементов. Здесь T – базовый тип элементов массива. Вершина стека соответствует целочисленной переменной ИНДЕКС. Тогда программа создания стека состоит в инициализации переменной ИНДЕКС нулём.

1. Программа СОЗД\_СТЕКА;

начало

ИНДЕКС:= 0;

конец;

Другие операции можно реализовать следующим образом.

2. Программа ДОБАВЛЕНИЕ (аргумент t:T; модифицируемые параметры СТЕК [1..n ]:T; ИНДЕКС: ЦЕЛОЕ);

начало

если ИНДЕКС=n то ОШИБКА

иначе

начало ИНДЕКС:= ИНДЕКС+1;

СТЕК [ИНДЕКС]:= t

конец

конец;

3. Программа УДАЛЕНИЕ (аргумент t:T; модифицируемые параметры СТЕК [1..n ]:T; ИНДЕКС: ЦЕЛОЕ);

начало

если ИНДЕКС = 0 то ОШИБКА

иначе ИНДЕКС = ИНДЕКС - 1

конец;

При сплошном представлении легче адресация, но требуется заранее указывать размер массива, следовательно, либо не используется часть массива (резерв), либо может возникнуть переполнение, если размер массива мал.

Тема: ИЗУЧЕНИЕ СПОСОБОВ ФИЗИЧЕСКОГО ПРЕДСТАВЛЕНИЯ ОЧЕРЕДИ.

ОЧЕРЕДЬ — динамическая структура данных, организованная по принципу «Первым пришёл — первым ушёл». Другое название очереди — структура FIFO (это название происходит от англоязычного аналога указанного принципа: «First In — First Out»).

Очередь характеризуется:

* **последовательным доступом** к элементам, организованным в ОЧЕРЕДЬ;
* **неразрушающей записью** (при добавлении новых данных в структуру);
* **двумя точками доступа** к данным, образующим ОЧЕРЕДЬ (одна точка доступа, называемая "хвостом", используется для пополнения ОЧЕРЕДИ, другая, называемая "головой", служит для удаления элемента из ОЧЕРЕДИ).

Допустимыми операциями при работе с ОЧЕРЕДЬЮ в классическом варианте считаются следующие:

1. создание ОЧЕРЕДИ:
2. проверка, является ли ОЧЕРЕДЬ пустой;
3. добавление нового элемента в ОЧЕРЕДЬ;
4. удаление элемента, находящегося в "голове";
5. просмотр элемента, находящегося в "голове".

При практической реализации очереди к этим операциям обычно добавляют возможность просмотра содержимого всей структуры данных и возможность очистки памяти, выделенной под очередь, когда потребность в её использовании в программе отпала. Особенностью операции удаления является то, что не требуется указывать удаляемый элемент, так как всегда удаляется элемент, находящийся в "голове" очереди.

Структуры типа "очередь" используют:

* при рассмотрении задач массового обслуживания (телефонная станция, станция "скорой помощи" и т.п.);
* при решении проблем функционирования операционной системы компьютера по обслуживанию многозадачного и многопользовательского режимов.

### **Физическое представление очереди**

Принято рассматривать два основных способа физической реализации структуры данных ОЧЕРЕДЬ:

1. **Сплошное представление** – соответствующее отображению элементов ОЧЕРЕДИ на одномерный массив.
2. **Цепное** **представление** – в динамической области памяти, с использованием указателей.

Оба варианта требуют наличия двух точек входа (голова и хвост).

При сплошном представлении особого внимания требует ситуация переполнения массива при добавлении элементов. Пустая очередь при любом физическом представлении приводит к ошибке при попытке бесконтрольного применения к ней операции удаления.

При реализации сплошного представления очереди она отображается на одномерный массив, и вводятся две переменные, имеющие тип индекса массива, которые исполняют роль указателей на "голову" и "хвост".

1

2

3

4

5

6

7

очередь

голова

При сплошном представлении легче адресация, но требуется заранее указывать размер массива, следовательно, либо не используется часть массива (резерв), либо может возникнуть переполнение, если размер массива мал.

**Замечание**. При представлении очереди удобно предполагать, что указатель "хвост" установлен на первый свободный элемент массива, куда можно добавить элемент очереди. В этом случае облегчается проверка переполнения массива.

### **1.1.1. Варианты работы со сплошным представлением очереди**

1. После удаления элемента из головы все элементы очереди сдвигаются к началу массива. В качестве иллюстрации можно рассмотреть перемещение магнитной ленты над неподвижной считывающей головкой. Недостаток: необходимость перемещения большого числа элементов.
2. Сдвиг производится, если хвост достиг границы массива, то есть нет места для размещения новых данных в очереди, хотя в начале массива есть свободное место. Можно считать, что магнитная лента не перемещается, а передвигается считывающая магнитная головка.
3. Закольцевать массив – вообще отсутствует операция по перезаписи очереди внутри массива размещения. В этом случае перемещаются лишь считывающая и записывающая головки.

### **1.1.2. Сплошное представление очереди**

Пусть T ⎯ тип данных, помещаемых в очередь (базовый тип элементов очереди). Тогда для реализации сплошного представления очереди требуется выполнить следующие описания:

1. Массив ОЧЕРЕДЬ [1:n]:T;
2. Переменные хвост (j), голова (i): ЦЕЛЫЕ.

Операции с очередью реализуются в виде подпрограмм, работающих с введенным массивом ОЧЕРЕДЬ, "указателем" i, соответствующим "голове", и "указателем" j, соответствующим "хвосту":

1. При добавлении элемента в хвост очереди переменная j изменяется на 1 (+1 до достижения n, а затем производится переход на 1 элемент массива);
2. При удалении элемента из головы i изменяется на 1 (+1 до достижения n, а затем переход на 1).
3. Очередь пуста ⎯ соответствует i = j после выполнения операции удаления.
4. Массив полон ⎯ соответствует i = j после выполнения операции пополнения очереди.

Если ввести дополнительный элемент массива с индексом 0, то в нём можно хранить число элементов очереди. Тогда упрощаются проверки переполнения массива и случая пустой очереди. При этом алгоритмы операций с очередью можно представить в виде:

1. Программа Созд\_очереди (арг массив ОЧЕРЕДЬ[0:n] : T; голова, хвост : ЦЕЛЫЕ);

начало

голова := 1 ;

хвост := 1 ;

ОЧЕРЕДЬ[0] := 0

конец;

1. Программа Добавить\_элемент (арг : элем :T; массив ОЧЕРЕДЬ[0:n] : T; голова, хвост :ЦЕЛЫЕ);

начало

если ОЧЕРЕДЬ[0] = n то ОШИБКА

иначе начало ОЧЕРЕДЬ [хвост] := элем;

хвост:=хвост+1;

ОЧЕРЕДЬ[0] := ОЧЕРЕДЬ[0]+1;

конец;

если хвост = n+1 то хвост := 1;

конец;

1. Программа Удалить (арг массив ОЧЕРЕДЬ[0:n] : T; голова, хвост : ЦЕЛЫЕ);

начало

если ОЧЕРЕДЬ[0] = 0 то ОШИБКА

иначе начало голова := голова+1;

ОЧЕРЕДЬ[0] := ОЧЕРЕДЬ[0]-1;

конец;

если голова = n+1 то голова := 1;

конец;

4. Программа Первый (арг массив ОЧЕРЕДЬ[0..n]:T; голова, хвост : ЦЕЛЫЕ)

начало

если ОЧЕРЕДЬ[0] = 0 то ОШИБКА

иначе ПЕРВЫЙ:=ОЧЕРЕДЬ [голова]

конец ;

Замечание 1. ОШИБКА соответствует описанию действий, которые должны быть выполнены.

Замечание 2. В рассмотренных алгоритмах по умолчанию предполагается для простоты, что тип T соответствует типу ЦЕЛЫХ. В других случаях служебная информация о фактической длине очереди может храниться отдельно от массива размещения данных.

* + 1. Цепное представление очереди

Под цепным представлением понимают представление в динамически распределяемой области памяти (в «куче») в виде цепочки элементов, связанных с помощью указателей. В качестве точки входа в цепочку применяют указатель на голову. Описание типа соответствующего указателя на Турбо Паскале может быть дано в форме:

Type UKAZ=^OCHER;

OCHER=record

info: integer;

next: UKAZ {указатель на следующий элемент цепочки}

end;

Var H, E, NE, KON: UKAZ; value: integer;

Здесь указатель включён в структуру данных. Поле записи info имеет тип integer. Это означает, что в рассматриваемом примере тип данных, образующих ОЧЕРЕДЬ, предполагается целочисленным. В случае другого типа данных поле info следует изменить.

Далее рассматривается вариант реализации операций с ОЧЕРЕДЬЮ при цепном представлении. При этом использованы процедуры и функция (реализация на языке Турбо Паскаль).

*1. Создание очереди (H- указатель на голову, E- указатель на хвост)*

Procedure Sozd\_Och;

BEGIN

E := Nil;

##### H := E

END;

*Замечание.* На практике часто операцию создания очереди совмещают с внесением в неё начальных данных.

*2. Добавление элемента в очередь (Добавляемый элемент – value)*

Procedure Dobav\_Och;

BEGIN

New(NE);

NE^.next:=nil;

NE^.info:= value;

If H=nil then H:=NE {очередь пуста}

else E^.next:=NE; {в конец очереди}

**E := NE** {Е – указатель на последний}

END;

*3. Удаление элемента из очереди*

Procedure Udal\_Och;

**var KP : Ukaz;**

**BEGIN**

**if H = Nil then**

**begin**

**writeln(‘пустая очередь – удалить нельзя’);**

**exit**

**end**

**else**

**begin**

KP := H; {Для сохранения доступа к фрагменту памяти, занимаемому удаляемым элементом. }

**H := H^.next;**

**dispose(H)** {Возврат памяти в «кучу»}

**end**

**END;**

*4. Чтение первого элемента в очереди*

Function First (H:UKAZ; Var ERR: boolean): integer;

BEGIN

If H=Nil then begin

Writeln(‘очередь пуста’); err:=true;

Exit

end

else begin {чтение успешно}

write(‘первый=’, H^.info:4);

err:=false;

First:= H^.info

end

END;

*5. Просмотр элементов очереди*

Procedure Cont\_Och;

BEGIN

Kon:= H;

while Kon<>Nil do

begin

writeln (Kon^.info);

Kon :=Kon^.next

end

END;

ЗАДАНИЕ

1. Разработать подпрограммы операций функциональной спецификации стека при сплошном и цепном представлении (включая проверку на пустоту и просмотр содержимого стека).
2. Разработать демонстрационную программу работы со стеком и просмотра результатов операций.
3. Разработать подпрограммы операций функциональной спецификации очереди (включая проверку на пустоту и просмотр содержимого очереди).
4. Разработать процедуру выборки-удаления элемента из очереди, совмещающей взятие первого элемента для обработки с удалением его из очереди.
5. Разработать демонстрационную программу работы с очередью и просмотра результатов операций.