Абстрактные структуры данных

Структуры данных и алгоритмы – это те материалы, из которых строятся программы.

Под СТРУКТУРОЙ ДАННЫХ в общем случае понимают множество элементов данных и множество связей между ними.

Классификация структур данных:

Абстрактные структуры данных

Базовые структуры

Базовыми структуры данных называют также примитивными или простыми структурами. Эти структуры служат основой для построения более сложных структур. В языках программирования простые структуры описываются простыми (базовыми) типами. К таким типам относятся: **числовые, битовые, логические, символьные, перечисляемые, интервальные, указатели.**

- **Числовые данные –** с помощью **целых чисел** может быть представлено количество объектов, являющихся дискретными по своей природе (т.е. счетное число объектов); значение **вещественных чисел** определяется лишь с некоторой конечной точностью, зависящей от внутримашинного формата вещественного числа.
- **Битовые –** для работы с отдельными двоичными разрядами данных.
- **Логические –**данные в виде одной из констант false (ложь) или true (истина).
- **Символьные –** символы из некоторого предопределенного множества. В большинстве современных ПЭВМ этим множеством является ASCII (American Standard Code for Information Interchange - американский стандартный код для обмена информацией).
- **Перечисляемые –** упорядоченные данные, определяемый программистом, т.е. программист перечисляет все значения, которые могут принимать эти данные.
- **Интервальные –** ограничение допустимого диапазона значений некоторых стандартных скалярных данных или рамок описанных перечислимых данных.
- **Указатели –** адрес ячейки памяти (в подавляющем большинстве современных вычислительных систем размер ячейки - минимальной адресуемой единицы памяти - составляет один байт).

Абстрактные структуры данных

Статические структуры

Статические структуры относятся к разряду непримитивных структур, которые, фактически, представляют собой структурированное множество примитивных, базовых, структур. Статические структуры отличаются отсутствием изменчивости, память для них выделяется автоматически - на этапе компиляции или при выполнении (в момент активизации того программного блока, в котором они описаны).

- **Векторы** (одномерные массивы) – структуры данных с фиксированным числом элементов одного и того же типа. Каждый элемент вектора имеет уникальный в рамках заданного вектора номер. Обращение к элементу вектора выполняется по имени вектора и номеру требуемого элемента.
- **Массивы** – структура данных, которая характеризуется: фиксированным набором элементов одного и того же типа; каждый элемент имеет уникальный набор значений индексов; количество индексов определяют мерность массива, например, три индекса - трехмерный массив, один индекс - одномерный массив или вектор; обращение к элементу массива выполняется по имени массива и значениям индексов для данного элемента.

  Иначе: массив - это вектор, каждый элемент которого - вектор.
- **Множества** – это набор неповторяющихся данных одного и того же типа.
- **Записи (структуры)** – конечное упорядоченное множество полей, характеризующихся различным типом данных (базовыми и статическими).
- **Таблицы** – представляет собой вектор, элементами которого являются записи. Характерная логическая особенность таблиц – доступ к элементам таблицы производится не по номеру (индексу), а по ключу - по значению одного из свойств объекта, описываемого записью - элементом таблицы. Ключ - это свойство, идентифицирующее данную запись во множестве однотипных записей. Как правило, к ключу предъявляется требование уникальности в данной таблице. Ключ может включаться в состав записи и быть одним из ее полей, но может и не включаться в запись, а вычисляться по положению записи. Таблица может иметь один или несколько ключей.

Абстрактные структуры данных

Полустатические структуры

Полустатические структуры данных характеризуются следующими признаками: **(1) они имеют переменную длину и простые процедуры ее изменения;** **(2) изменение длины структуры происходит в определенных пределах, не превышая какого-то максимального (предельного) значения. Доступ к элементу *может* осуществляться по его порядковому номеру.**

- **Строки – это линейно упорядоченная последовательность символов, принадлежащих конечному множеству символов, называемому алфавитом. Их важные свойства:**
  - **длина, как правило, переменна, хотя алфавит фиксирован;**
  - **обычно обращение к символам строки идет с какого-нибудь одного конца последовательности, т.е. важна упорядоченность этой последовательности, а не ее индексация; в связи с этим свойством строки часто называют также цепочками;**
  - **чаще всего целью доступа к строке является не отдельный ее элемент (хотя это тоже не исключается), а некоторая цепочка символов в строке.**
- **Стеки – последовательный список переменной длины, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого вершиной стека. Применяются и другие названия стека - магазин и очередь, функционирующая по принципу LIFO: Last – In, First- Out - "последним пришел - первым исключается (выбирается)".**
- **Очереди – последовательный (линейный) список переменной длины, в котором включение элементов выполняется только с одной стороны списка (эту сторону часто называют концом или хвостом очереди), а исключение - с другой стороны (называемой началом или головой очереди). Очередью работает по схеме FIFO (First – In, First- Out - "первым пришел - первым исключается") или по схеме приоритетов – порядок исключения зависит от приоритета элемента очереди.**
- **Деки – особый вид очереди, от англ. deq - double ended queue, т.е. очередь с двумя концами - это последовательный список, в котором как включение, так и исключение элементов может осуществляться с любого из двух концов списка. Структура дека аналогична структуре кольцевой FIFO-очереди. Но применительно к деку надо говорить не о начале и конце, а о левом и правом конце.**

Абстрактные структуры данных

Динамические структуры

Динамические структуры характерны отсутствием физической смежности элементов структуры в памяти и непостоянством и непредсказуемостью размера (числа элементов) структуры в процессе ее обработки.

Связное представление данных

Элементы динамической структуры располагаются по непредсказуемым адресам памяти, поэтому адрес элемента не может быть вычислен из адреса начального или предыдущего элемента. Для установления связи между элементами динамической структуры используются специальные указатели, через которые устанавливаются *явные* связи между элементами. Такое представление данных в памяти называется связным. Элемент динамической структуры состоит из двух полей:

- **информационного поля или поля данных, в котором содержатся те данные, ради которых и создается структура; в общем случае информационное поле само является интегрированной структурой - вектором, массивом, записью и т.п.;**
- **поле связок, в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры.**

Достоинства связного представления данных - в возможности обеспечения значительной изменчивости структур (размер структуры ограничивается только доступным объемом машинной памяти; при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей).

Недостатки связного представления:

- **работа с указателями требует, как правило, более высокой квалификации от программиста;**
- **на поля связок расходуется дополнительная память;**
- **доступ к элементам связной структуры может быть менее эффективным по времени.**

Последний недостаток является наиболее серьезным и именно им ограничивается применимость связного представления данных. Если в смежном представлении данных для вычисления адреса любого элемента нам во всех случаях достаточно было номера элемента и информации, содержащейся в дескрипторе структуры, то для связного представления адрес элемента не может быть вычислен из исходных данных. Дескриптор связной структуры содержит один или несколько указателей, позволяющих войти в структуру, далее поиск требуемого элемента выполняется следованием по цепочке указателей от элемента к элементу. *Поэтому связное представление практически никогда не применяется в задачах, где логическая структура данных имеет вид вектора или массива - с доступом по номеру элемента, но часто применяется в задачах, где логическая структура требует другой исходной информации доступа (таблицы, списки, деревья и т.д.).*

Абстрактные структуры данных

Динамические структуры

- **Линейные связанные списки –** упорядоченное множество, состоящее из переменного числа элементов, отражающий отношения соседства между элементами. Односвязный список имеет в поле связок каждого элемента один указатель: следующий элемент. Кроме того, есть особые указатели: начало списка и конец списка. Двухсвязный список имеет в поле связок два указателя: следующий элемент и предыдущий элемент. Можно задать кольцевой список – из последнего элемента указатель ссылается на первый элемент. Многосвязный список (мультисписок) – каждый элемент включает несколько указателей на связи между собой подмножеств данного списка.
- **Разветвлённые связанные списки (Нелинейные разветвленные списки) –** это списки, элементами которых могут быть тоже списки. Выше рассмотрены двухсвязные линейные списки. Если один из указателей каждого элемента списка задает порядок обратный к порядку, устанавливаемому другим указателем, то такой двусвязный список будет линейным. Если же один из указателей задает порядок произвольного вида, не являющийся обратным по отношению к порядку, устанавливаемому другим указателем, то такой список будет нелинейным.
- **Графы –** это сложная нелинейная многосвязная динамическая структура, отображающая свойства и связи сложного объекта. Её свойства:
  1) на каждый элемент (узел, вершину) может быть произвольное количество ссылок;
  2) каждый элемент может иметь связь с любым количеством других элементов;
  3) каждая связка (ребро, дуга) может иметь направление и вес.

  В узлах графа содержится информация об элементах объекта. Связи между узлами задаются ребрами графа. Ребра графа могут иметь направленность, показываемую стрелками, тогда они называются ориентированными графами (орграф), ребра без стрелок - неориентированные.
- **Деревья –** это графы, которые характеризуются следующими свойствами:
  1. Существует единственный элемент (узел или вершина), на который не ссылается никакой другой элемент - и который называется КОРНЕМ;
  2. Начиная с корня и следуя по определенной цепочке указателей, содержащихся в элементах, можно осуществить доступ к любому элементу структуры;
  3. На каждый элемент, кроме корня, имеется единственная ссылка, т.е. каждый элемент адресуется единственным указателем.

ВЕТВИ – ребра графа-дерева, ЛИСТЬЯ – конечные узлы дерева; ЛЕС – несколько различных деревьев. Бинарные деревья.

Абстрактные структуры данных

Файловые структуры

Файловые структуры данных – это сами файлы, их внутренняя организация и способы доступа к их содержимому.

- **Файловая структура последовательного доступа – это набор последовательных элементов, ввод которых в файл происходит в порядке их поступления от программы (устройства), а вывод из файла в порядке их расположения в файле.**
- **Файловая структура прямого доступа – это набор элементов, занимающих последовательные позиции в линейном порядке; значение может быть передано в элемент файла (или из него), находящийся в любой выбранной позиции. Позиция элемента задается его индексом (ключом).**
- **Файловая структура комбинированного доступа – это набор элементов, которые допускают как прямой доступ по индексу (ключу), так и последовательный в порядке возрастания индексов (ключей).**

Структуры данных

Стек (stack)

Стеки – последовательный список переменной длины, включение и исключение элементов из которого выполняются только с одной стороны списка, называемого вершиной стека. Применяются и другие названия стека - магазин и очередь, функционирующая по принципу LIFO: Last – In, First- Out - "последним пришел - первым исключается (выбирается)".

Основные операции над стеком:

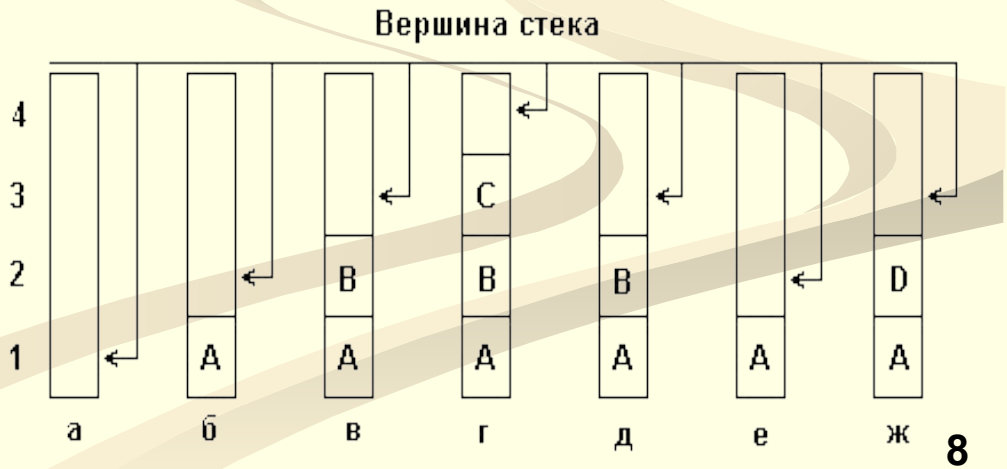
- включение нового элемента (английское название push - заталкивать)
- исключение элемента из стека (англ. pop - выскакивать).

Полезными могут быть также вспомогательные операции:

- **определение текущего числа элементов в стеке;**
- **очистка стека;**
- **неразрушающее чтение элемента из вершины стека, которое может быть реализовано, как комбинация основных операций: x:=pop(stack); push(stack,x).**

Пример: принцип включения элементов в стек и исключения элементов из стека.

На рисунке изображены состояния стека:
а) пустого;
б-г) после последовательного включения в него элементов с именами 'A', 'B', 'C';
д, е) после последовательного удаления из стека элементов 'C' и 'B';
ж) после включения в стек элемента 'D'.



Структуры данных

C / C++

Стек (stack)

Через массив

```
int stack [MAX];
int tos=0;  // вершина стека

void push (int i)
/* Занесение элемента в стек */
{
if (tos >= MAX)
    {
      printf (" Стек полон \n");
      return 0;
      }
   // заносим элемент в стек
stack[tos] = i;
      tos++;
return 0;
}
```

```
int pop (void)
/* Выборка элемента из стека  */
tos--;
if (tos < 0)
   {
      printf (" Стек пуст \n");
      return 0;
      }
  return stack [tos];
}
```

Написать головную программу:

1. Занести в стек 5 элементов
2. Вывести содержимое стека на экран
3. Удалить верхний элемент
4. Удалить второй снизу элемент
5. Вывести оставшуюся часть стека

Классы и объекты

Работа со СТКЕом в ООП через массив

```
#include <iostream.h>
#define SIZE 100
// Определение класса stack
class stack
{
  int stck[SIZE];
  int tos;
 public:
   void init ();
   void push (int i);
   int pop ();
};
//-----------------------------------------------
void stack :: init () /* :: - оператор разрешения области видимости (доступ к области видимости). Он определяет компилятору, что данная версия функции init() принадлежит классу stack, т.е. находится в области видимости этого класса */
{  tos = 0;  }
//-----------------------------------------------
void stack :: push (int i)
{ if (tos == SIZE)
    { cout << "Стек полон.\n"; return;  }
   stck[tos] = i;
   tos++;
}                    // см. продолжение
```

```
                    // продолжение
int stack :: pop ()
{ if (tos == 0)
   { cout << "Стек пуст.\n"; return 0;  }
  tos--;
  return stck[tos];  }
//-----------------------------------------------
int main ()
{   stack    stack1, stack2; /* создаем 2 объекта класса stack */
    stack1.init ();   // Вызов init для объекта stack1
    stack2.init ();   // Вызов init для объекта stack2
  stack1.push (1);
  stack2.push (2);
     stack1.push (3);
     stack2.push (4);

 cout << stack1.pop() <<  "  ";
 cout << stack1.pop() <<  "  ";
 cout << stack2.pop() <<  "  ";
 cout << stack2.pop() <<  " \n ";
return 0;
}
```

Вывод на экран: 3 1 4 2

Структуры данных

C / C++

Работа со СТЕКом в ООП
через указатели и элементы односвязного списка

```
#include <iostream>
  struct stek       // Описание элемента стека
    {   int value; struct stek *next; /* значение, следующий элемент */   };

void push(stek* &NEXT, const int VALUE)  //  Добавить элемент в стек
  {   stek *MyStack = new stek;
    MyStack->value = VALUE;
    MyStack->next = NEXT;
    NEXT = MyStack;
  }

int pop(stek* &NEXT)  //  Удалить элемент из стека
   {   int temp = NEXT->value;
       stek *MyStack = NEXT;
       NEXT = NEXT->next;
     delete MyStack;
      return temp;
    }
int main()
  {   stek *p=0;
            push(p,100);
            push(p,200);
       cout << pop(p);
       cout << pop(p);
    return 0;
  }
```

Структуры данных

Очереди (queue)

Очереди – последовательный (линейный) список переменной длины, в котором **добавление** элементов выполняется только с одной стороны списка (эту сторону часто называют **концом** или хвостом **очереди**), а **выборка** - с другой стороны (называемой **началом** или головой **очереди**). Очередью работает по схеме **FIFO** (First – In, First- Out - "первым пришел - первым ушел") или по схеме **приоритетов** – порядок выборки (ухода) зависит от приоритета элемента очереди.

Основные операции над очередями:

- добавление нового элемента;
- выборка элемента из очереди ;
- определение текущего числа элементов в очереди;
- очистка очереди;
- неразрушающее чтение элемента.

Пример: принцип работы очереди с использование функций qin – поставить в очередь и qout - выбрать из очереди

Функция	Действие	Содержимое очереди (с головы)
qin (A)	поставить A в очередь	A
qin (B)	поставить B в очередь	A B
qin (C)	поставить C в очередь	A B C
qout ()	выбрать A из очереди	B C
qin (D)	поставить D в очередь	B C D
qout ()	выбрать B из очереди	C D
qout ()	выбрать C из очереди	D

Структуры данных

Очереди (queue)

C / C++

```
#include <iostream.h>
  struct Node {   int d;   Node *p;  };

Node * firstq (int d)
// Начальное формирование очереди
{
  Node *pv = new Node;
   pv -> d  = d;
   pv -> p  = 0;
 return pv;
}

void addq (Node **pend, int d)
// Добавление элемента  в конец очереди
{
   Node *pv = new Node;
   pv -> d  = d;
   pv -> p  =0;
   (*pend) -> p  = pv;
   *pend = pv;
}

int delq (Node **pbeg)
// Выборка элемента из очереди
{
int temp = (*pbeg) -> d;
  Node *pv = *pbeg;
  *pbeg = (*pbeg) -> p;
  delete pv;
  return temp;
}
```

Сформировать очередь из N целых чисел и вывести её на экран. Функции разместить в конце программы.

```
#include <iostream.h>
  struct Node { int d; Node *p;  };   int N;

Node * firstq (int d);
void addq (Node **pend, int d);
int delq (Node **pbeg);


int main()
{  Node *pbeg = firstq(1);
 Node *pend = pbeg;
cout << "Введите размер очереди - "; cin >> N;
 for (int i=2; i<N+1; i++)
    addq (&pend, i);
 while (pbeg)
   cout << delq (&pbeg) << '  ';
   cout << "\n ";
 return 0;  }

//Функции работы с очередью
.....................................
```

Вопрос:

Объяснить работу указателей в этих функциях. В частности, объяснить конструкции:

Node **pend и int delq (Node **pbeg);