|  |  |  |
| --- | --- | --- |
| Структура данных (СД) — общее свойство информационного объекта, с которым взаимодействует та или иная программа. Это общее свойство характеризуется: множеством допустимых значений данной структуры;  p1 ≤ i1 ≤ q1 pn ≤ in ≤ qn Adr(B[i1, i2, … , in]) = Adr(B[p1, p2, … , pn]) − S∑pkDk+∑ikDk  Dk зависит от способа размещения массива.  При размещении по строкам:  Dk = (qk+1 − pk+1 + 1)Dk+1, k = n – 1, n – 2, …., 1  При размещении по столбцам:  Dk = (qk−1 − pk−1 + 1)Dk−1, k = 2, 3, …, n | Таблица — это набор элементов одинаковой организации, каждый из которых можно представить в виде двойки , где K — ключ, а V — тело (информационная часть) элемента. Ключ уникален для каждого элемента, т.е. в таблице нет двух элементов с одинаковыми ключами. Ключ используется для доступа к элементам при выполнении операций. Таблица — динамическая структура. Над таблицей определены следующие основные операции: 1. Инициализация. 2. Включение элемента. 3. Исключение элемента с заданным ключем. 4. Чтение элемента с заданным ключем. 5. Изменение элемента с заданным ключем. 6. Проверка пустоты таблицы. 7. Уничтожение таблицы.  Алгоритм сортировки Шелла  1. Определить количество проходов t и шаг сортировки на каждом  проходе. Результат сохранить в массиве h.  2. На i-ом проходе (i=1,…,t) выполнить сортировку включением с  шагом h(i). | Над стеком определены следующие основные операции: 1. Инициализация. 2. Включение элемента. 3. Исключение элемента. 4. Чтение элемента. 5.  Проверка пустоты стека. 6. Уничтожение стека  Интерфейс СД типа стек на массиве  #define Stack\_Size 100 const int StackOk=0; const int StackEmpty=1; const int StackFull=2;extern int StackError=0;  typedef int BaseType;  typedef struct { BaseType buf [Stack\_Size]; unsigned uk; } Stack;  void StackInit(Stack \*s); int StackisEmpty(Stack \*s); int StackisFull(Stack \*s); void StackPut(Stack \*s, BaseType e);  void StackGet (Stack \*s, BaseType \*e); |
| Очередь — это динамическая структура.  Над очередью определены следующие основные операции: 1. Инициализация. 2. Включение элемента. 3. Исключение элемента. 6. Проверка пустоты  очереди. 7. Уничтожение очереди.  Интерфейс СД типа очередь const int FifoOk = 0; const int FifoFull = 1;const int FifoEmpty = 2; extern int errorFifo;typedef int BaseType; typedef struct  { BaseType buf[SIZE\_FIFO]; unsigned ukEnd; unsigned ukBegin; unsigned len;} FIFO;  void initFifo(FIFO \*F); void putFifo(FIFO \*F, BaseType E); void geFifo(FIFO \*F, BaseType \*E); int FifoisFull(FIFO \*F);  int FifoisEmpty(FIFO \*F); | Над СД ЛС определены следующие основные операции: 1. Инициализация. 2. Включение элемента. 3. Исключение элемента.4. Чтение текущего элемента. 5. Переход в начало списка. 6. Переход в конец списка. 7. Переход к следующему элементу. 8. Переход к i-му элементу. 9. Определение длины списка. 10. Уничтожение списка  #ifndef LIST\_H\_INCLUDED #define LIST\_H\_INCLUDED  static const int ListOk=0, ListNotMem=1, ListEmpty=2, ListEnd=3;static short ListError; //переменная ошибок  typedef /\*базовый тип\*/ BaseType; typedef struct element \*ptrEl; typedef struct element{ BaseType data; ptrEl next;};  typedef struct List { ptrEl start; ptrEl ptr;};  void InitList(List \*L); void PutList(List \*L, BaseType E); void GetList(List \*L, BaseType \*E); //Исключение элемента из списка  int EmptyList(List \*L); int EndList(List \*L); void MovePtr(List \*L); void DoneList(List \*L); //Удаление списка (деструктор)  #endif // LIST\_H\_INCLUDED | Стек как отображение на список. Очередь как  отображение на список  Стек:#ifndef STACONOLS\_H\_INCLUDED #define STACONOLS\_H\_INCLUDED #include "List.h" #define StackOk ListOk #define StackNotMem ListNotMem #define StackEnd ListEnd short StackError;typedef List Stack; void StackInit(Stack \*S); void StackDone(Stack \*S); void StackPut(Stack \*S, BaseType E); void StackGet(Stack \*S, BaseType \*E); int StackEmpty(Stack \*S); #endif // STACONOLS\_H\_INCLUDED Пример: void StackInit(Stack \*S) {InitList(S);StackError = ListError;}  Очередь:#ifndef FIFOOLS\_H\_INCLUDED #define FIFOOLS\_H\_INCLUDED #include "List.h" #define FifoOlsOk ListOk #define FifoOlsNotMem ListNotMem #define FifoEmpty ListUnder short FifotError; typedef List FifoOls; typedef BaseType BaseTypeFifo; void initFifoOls(FifoOls \*FO); void putFifoOls(FifoOls \*FO, BaseTypeFifo E); void getFifoOls(FifoOls \*FO, BaseTypeFifo \*E); void doneFifoOls(FifoOls \*FO); int EmptyFifoOls(FifoOls \*FO); #endif // FIFOOLS\_H\_INCLUDED |
| ДЕК Операции: Инициализация Включение элемента в начало / конец дека Исключение элемента из начала / конца  дека Проверка: дек пуст / не пуст Удаление дека  Обычно для реализации дека используют двусвязный линейный список, то есть на основе готовых функций списка  создают новые для дека.  static const int DackOk=0, DackEmpty=1, DackNotMem=2; static int DackError; //переменная ошибок  typedef t\_dlist Dack; //дек отображается на ДЛС, базовый тип задается в реализации ДЛС  void InitDack(Dack \*D); void DoneDack(Dack \*D);  void PutDackB(Dack \*D, t\_base E); void PutDackE(Dack \*D, t\_base E); void GetDackB(Dack \*D, t\_base \*E); void GetDackE(Dack  \*D, t\_base \*E); void ReadDackB(Dack \*D, t\_base \*E); void ReadDackE(Dack \*D, t\_base \*E); int EmptyDack(Dack \*D); | 1 класс. Задачи класса Р – это задачи, для которых известен алгоритм и временная сложность оценивается полиномиальной функцией f(N) = a k\*N + a k-1\*N 2 +…+a 0\*N k+1. Алгоритмы таких задач называют также “хорошими” алгоритмами. Этих алгоритмов мало, они образуют небольшой по мощности класс. Легко реализуются. Функция временной сложности для таких алгоритмов имеет вид O(Nk). 2 класс. Задачи класса Е – это задачи, для которых алгоритм известен, но сложность таких алгоритмов O(f N), где f – константа. Задачи такого класса – это задачи построения всех подмножеств некоторого множества, задачи построения всех полных подграфов графа. При небольших N экспоненциальный алгоритм может работать быстрее полиномиального. 3 класс. На практике существуют задачи, которые не могут быть отнесены ни к одному из вышерассмотренных классов. Это прежде всего задачи, связанные с решением систем уравнений с целыми переменными, задачи определения цикла, проходящего через все вершины некоторого графа, задачи диагностики и т.д. Такие задачи независимы от компьютера, от языка программирования, но могут решаться человеком. | БД – динамическая структура. Над СД БД определены следующие основные операции: инициализация, создание корня, запись данных, чтение данных, проверка – есть ли левый/правый сын, переход к левому/правому сыну, проверка, пустое ли дерево, удаление листа.  #define SizeMem 100 #define TreeOk 0 #define TreeNotMem 1  #define TreeUnder 2 extern int TreeError; typedef int BaseType;  typedef unsigned char PtrEl; typedef struct element{ BaseType Data; PtrEl LSon; PtrEl RSon; }element typedef PtrEl \*Tree;  element MemTree[SizeMem]; void InitTree(Tree \*T); void CreateRoot(Tree \*T); void WriteDataTree(Tree \*T, BaseType E);  void ReadDataTree(Tree \*T, BaseType \*E); int IsLSon(Tree \*T);  int IsRSon(Tree \*T); void MoveToLSon(Tree \*T, Tree \*TS);  void MoveToRSon(Tree \*T, Tree \*TS) int IsEmptyTree(Tree \*T);  void DellTree(Tree \*T); |