Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №8 по теме: «Структура данных „таблица“»

Выполнил:  
студент группы ПВ-21  
Адаменко И. И.

Проверил:  
профессор  
Синюк В. Г.

Белгород  
2013

Цель работы: изучить структуры данных типа «таблица», научиться их программно реализовывать и использовать.

## Задания

1. Для СД типа «таблица» определить:
   1. Абстрактный уровень представления:
      1. Характер организованности и изменчивости.
      2. Набор допустимых операций.
   2. Физический уровень представления:
      1. Схему хранения.
      2. Объем памяти, занимаемый экземпляром СД.
      3. Формат внутреннего представления СД и его способ интерпретации.
      4. Характеристику допустимых значений.
      5. Тип доступа к элементам.
   3. Логический уровень представления СД. Способ описания СД и экземпляра СД на языке программирования.
2. Реализовать СД «таблица» в соответствии с вариантом индивидуального задания в виде модуля на языке C.
3. Разработать программу на языке C для решения задачи в соответствии с вариантом индивидуального задания с использованием модулей, полученных в результате выполнения п. 2.

### Задача

Текст программы на некотором алгоритмическом языке может содержать символы-разделители, служебные слова, числовые константы и идентификаторы (слова, начинающиеся не с цифры и не являющиеся служебными). Вывести все идентификаторы, которые встречаются в программе.

Исходные данные: текстовые файлы, содержащие:

1. текст программы;
2. символы-разделители;
3. служебные слова.

Для хранения символов-разделителей и служебных слов использовать таблицы.

### Модуль для реализации таблицы

Неупорядоченная таблица на последовательном линейном списке.

|  |  |
| --- | --- |
| 01 | **#ifndef** UNSORTED\_TABLE\_ARRAY\_H |
| 02 | **#define** UNSORTED\_TABLE\_ARRAY\_H |
| 03 |  |
| 04 | **#include** <stdio.h> |
| 05 | **#include** <stdlib.h> |
| 06 |  |
| 07 | **const** **short** TableSize = 1000; |
| 08 | **const** **short** TableOk = 0; |
| 09 | **const** **short** TableEmpty = 1; |
| 10 | **const** **short** TableFull = 2; |
| 11 | **const** **short** TableNotContent = 3; |
| 12 | **const** **short** TableAlreadyContent = 4; |
| 13 | **static** **short** TableError; |
| 14 |  |
| 15 | **typedef** **int** DataType; |
| 16 | **typedef** **char** \* KeyType; |
| 17 | **typedef** **struct** base { |
| 18 | DataType data; |
| 19 | KeyType key; |
| 20 | } base; |
| 21 |  |
| 22 | **typedef** **struct** { |
| 23 | base Buf[TableSize]; |
| 24 | **unsigned** Top; |
| 25 | } Table; |
| 26 |  |
| 27 | *// Инициализация таблицы* |
| 28 | **void** InitTable(Table \*T); |
| 29 |  |
| 30 | *// Вставка в таблицу* |
| 31 | **void** PutTable(Table \*T, DataType E, KeyType K); |
| 32 |  |
| 33 | *// Взятие из таблицы* |
| 34 | **void** GetTable(Table \*T, DataType \*E, KeyType K); |
| 35 |  |
| 36 | *// Чтение из таблицы* |
| 37 | **void** ReadTable(Table \*T, DataType \*E, KeyType K); |
| 38 |  |
| 39 | *// Проверка на пустоту* |
| 40 | **bool** EmptyTable(Table \*T); |
| 41 |  |
| 42 | *// Проверка на заполненность* |
| 43 | **bool** FullTable(Table \*T); |
| 44 |  |
| 45 | *// Возвращение кода ошибки* |
| 46 | **short** GetTableError(); |
| 47 |  |
| 48 | **#endif** |

## Характеристика СД типа «таблица»

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип является последовательным и динамическим.
   2. К типу применимы такие операции, как: инициализация, включение элемента, исключение элемента с заданным ключом и пр.
2. Физический уровень представления СД:
   1. Структура данных хранится последовательно или связно (в зависимости от реализации).
   2. Размер памяти, занимаемый каждым экземпляром СД зависит от максимального возможного количества элементов в нее входящих (и от выбранной схемы хранения отдельного элемента).
   3. В зависимости от схемы хранения, меняется и формат внутреннего представления СД.
   4. Допустимыми являются все значения типа данных, определенного как базовый.
   5. Тип доступа к элементам данной СД зависит от реализации.
3. На логическом уровне представления СД, то есть, на языке программирования С экземпляр СД описывается следующим образом: Table T;

## Текст модуля для реализации СД типа «таблица»

### UnsortedTableArray.h

|  |  |
| --- | --- |
| 01 | **#ifndef** UNSORTED\_TABLE\_ARRAY\_H |
| 02 | **#define** UNSORTED\_TABLE\_ARRAY\_H |
| 03 |  |
| 04 | **#include** <stdio.h> |
| 05 | **#include** <stdlib.h> |
| 06 |  |
| 07 | **const** **short** TableSize = 1000; |
| 08 | **const** **short** TableOk = 0; |
| 09 | **const** **short** TableEmpty = 1; |
| 10 | **const** **short** TableFull = 2; |
| 11 | **const** **short** TableNotContent = 3; |
| 12 | **const** **short** TableAlreadyContent = 4; |
| 13 | **static** **short** TableError; |
| 14 |  |
| 15 | **typedef** **int** DataType; |
| 16 | **typedef** **char** \* KeyType; |
| 17 | **typedef** **struct** base { |
| 18 | DataType data; |
| 19 | KeyType key; |
| 20 | } base; |
| 21 |  |
| 22 | **typedef** **struct** { |
| 23 | base Buf[TableSize]; |
| 24 | **unsigned** Top; |
| 25 | } Table; |
| 26 |  |
| 27 | *// Инициализация таблицы* |
| 28 | **void** InitTable(Table \*T); |
| 29 |  |
| 30 | *// Вставка в таблицу* |
| 31 | **void** PutTable(Table \*T, DataType E, KeyType K); |
| 32 |  |
| 33 | *// Взятие из таблицы* |
| 34 | **void** GetTable(Table \*T, DataType \*E, KeyType K); |
| 35 |  |
| 36 | *// Чтение из таблицы* |
| 37 | **void** ReadTable(Table \*T, DataType \*E, KeyType K); |
| 38 |  |
| 39 | *// Проверка на пустоту* |
| 40 | **bool** EmptyTable(Table \*T); |
| 41 |  |
| 42 | *// Проверка на заполненность* |
| 43 | **bool** FullTable(Table \*T); |
| 44 |  |
| 45 | *// Возвращение кода ошибки* |
| 46 | **short** GetTableError(); |
| 47 |  |
| 48 | **#endif** |

### UnsortedTableArray.cpp

|  |  |
| --- | --- |
| 01 | **#include** "UnsortedTableArray.h" |
| 02 |  |
| 03 | *// Инициализация таблицы* |
| 04 | **void** InitTable(Table \*T) { |
| 05 | T->Top = 0; |
| 06 | TableError = TableOk; |
| 07 | } |
| 08 |  |
| 09 | *// Вставка в таблицу* |
| 10 | **void** PutTable(Table \*T, DataType E, KeyType K) { |
| 11 | base tmp; |
| 12 | **if** (FullTable(T)) { |
| 13 | **return**; |
| 14 | } |
| 15 | ReadTable(T, &tmp.data, K); |
| 16 | **if** ((TableError == TableNotContent) || (TableError == TableEmpty)) { |
| 17 | tmp.data = E; |
| 18 | tmp.key = K; |
| 19 | T->Buf[T->Top++] = tmp; |
| 20 | TableError = TableOk; |
| 21 | } |
| 22 | **else** { |
| 23 | TableError = TableAlreadyContent; |
| 24 | } |
| 25 | } |
| 26 |  |
| 27 | *// Взятие из таблицы* |
| 28 | **void** GetTable(Table \*T, DataType \*E, KeyType K) { |
| 29 | **int** i = 0; |
| 30 | **if** (EmptyTable(T)) { |
| 31 | **return**; |
| 32 | } |
| 33 | **while** (i < T->Top) { |
| 34 | **if** (T->Buf[i].key == K) { |
| 35 | **int** j; |
| 36 | \*E = T->Buf[i].data; |
| 37 | **for** (j = i + 1; j < T->Top; j++) { |
| 38 | T->Buf[j - 1] = T->Buf[j]; |
| 39 | } |
| 40 | T->Top--; |
| 41 | **return**; |
| 42 | } |
| 43 | i++; |
| 44 | } |
| 45 | TableError = TableNotContent; |
| 46 | } |
| 47 |  |
| 48 | *// Чтение из таблицы* |
| 49 | **void** ReadTable(Table \*T, DataType \*E, KeyType K) { |
| 50 | **int** i = 0; |
| 51 | **if** (EmptyTable(T)) { |
| 52 | **return**; |
| 53 | } |
| 54 | **while** (i < T->Top) { |
| 55 | **if** (T->Buf[i].key == K) { |
| 56 | \*E = T->Buf[i].data; |
| 57 | **return**; |
| 58 | } |
| 59 | i++; |
| 60 | } |
| 61 | TableError = TableNotContent; |
| 62 | } |
| 63 |  |
| 64 | *// Проверка на пустоту* |
| 65 | **bool** EmptyTable(Table \*T) { |
| 66 | **if** (T->Top == 0) { |
| 67 | TableError = TableEmpty; |
| 68 | **return** true; |
| 69 | } |
| 70 | **return** false; |
| 71 | } |
| 72 |  |
| 73 | *// Проверка на заполненность* |
| 74 | **bool** FullTable(Table \*T) { |
| 75 | **if** (T->Top == TableSize) { |
| 76 | TableError = TableFull; |
| 77 | **return** true; |
| 78 | } |
| 79 | **return** false; |
| 80 | } |
| 81 |  |
| 82 | *// Возвращает код ошибки* |
| 83 | **short** GetTableError() { |
| 84 | **return** TableError; |
| 85 | } |

## Текст программы для решения задачи

### main.cpp

|  |  |
| --- | --- |
| 01 | **#define** \_CRT\_SECURE\_NO\_WARNINGS |
| 02 |  |
| 03 | **#include** <string.h> |
| 04 | **#include** "UnsortedTableArray.h" |
| 05 |  |
| 06 | **int** main() { |
| 07 | Table T; |
| 08 | **char** \* tmp = ""; |
| 09 | **char** \* tmpChar = ""; |
| 10 | **int** i; |
| 11 | **FILE** \* fcode; |
| 12 | **FILE** \* fspaces; |
| 13 | **FILE** \* fwords; |
| 14 |  |
| 15 | **if** (!(fcode = fopen(*"code.txt"*, *"r"*))) { |
| 16 | printf(*"Can't open code file!\n"*); |
| 17 | getchar(); |
| 18 | exit(1); |
| 19 | } |
| 20 | **if** (!(fspaces = fopen(*"spaces.txt"*, *"r"*))) { |
| 21 | printf(*"Can't open spaces file!\n"*); |
| 22 | getchar(); |
| 23 | exit(1); |
| 24 | } |
| 25 | **if** (!(fwords = fopen(*"words.txt"*, *"r"*))) { |
| 26 | printf(*"Can't open words file!\n"*); |
| 27 | getchar(); |
| 28 | exit(1); |
| 29 | } |
| 30 |  |
| 31 | InitTable(&T); |
| 32 |  |
| 33 | **while** (fscanf(fwords, *"%s"*, &tmp) == 1) { |
| 34 | PutTable(&T, NULL, tmp); |
| 35 | } |
| 36 |  |
| 37 | **while** (fscanf(fspaces, *"%c"*, &tmp) == 1) { |
| 38 | PutTable(&T, NULL, tmp); |
| 39 | } |
| 40 |  |
| 41 | tmp = ""; |
| 42 | **while** (fscanf(fcode, *"%c"*, &tmpChar) == 1) { |
| 43 | ReadTable(&T, &i, tmpChar); |
| 44 | **if** (GetTableError() != 0) { |
| 45 | strcat(tmp, tmpChar); |
| 46 | } |
| 47 | **else** { |
| 48 | tmp += '\0'; |
| 49 | ReadTable(&T, &i, tmp); |
| 50 | **if** ((tmp[0] <= '0') && (tmp[0] >= '9') && (GetTableError() != 0)) { |
| 51 | printf("%s ", tmp); |
| 52 | PutTable(&T, NULL, tmp); |
| 53 | } |
| 54 | tmp = ""; |
| 55 | } |
| 56 | } |
| 57 |  |
| 58 | getchar(); |
| 59 |  |
| 60 | **return** 0; |
| 61 | } |

## Результаты работы программы

|  |  |  |
| --- | --- | --- |
| № п/п | Исходные данные | Результат работы программы |
| Файлы | Сообщение на экране |
| 1 | *Файл с кодом программы* | lab\_2 Ax Ay Bx By Cx Cy AB BC AC Epsilon |
| program lab\_2;  var  Ax, Ay, Bx, By, Cx, Cy: real;  AB, BC, AC: real;  Epsilon: real;  begin  AB := sqr(Bx - Ax) + sqr(By - Ay);  BC := sqr(Cx - Bx) + sqr(Cy - By);  AC := sqr(Cx - Ax) + sqr(Cy - Ay);  end. |
| *Файл с символами-разделителями* |
| ;,-+():=. |
| *Файл со служебными словами* |
| program var begin end real |