НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Лабораторна робота №1

з дисципліни **«**Системне програмування – 2**»**

Виконав:

студент 3 курсу ФІОТ

гр. ІО-42

Кочетов Данило

Перевірив:

Павлов В. Г.

Київ – 2016 р.

## **Тема роботи**: **Модульне програмування в рамках базової системи проектування програм та його використання для побудови програм обробки таблиць**

Мета роботи: Вивчення типів таблиць в системних прог­ра­мах і конструкцій базової мови програмування для їх визначення. Пошук за прямою адресою. Основні типи залежностей та відно­шень, які реалізуються через пошук в таблицях системних програм. Лінійний та двійковий пошук. Вимоги до унікальності ключів.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № вар. | Тип ключа для прямої адреси | Тип ключа для інших видів пошуку | Тип функціо­нального поля | Тип вибірки |
| 13 | unsigned int | char\*\_ unsigned short | struct | Перший |

**Лістинг програми**

|  |  |
| --- | --- |
| **main.c**  #pragma once  #include "table.h"  int main() {  Table tb = (Table) malloc(100 \* sizeof(Record));  int size = 0;  printRec(insLinear(rec("Diatlov", 88, 4), tb, &size));  printRec(insLinear(rec("Kutsar", 89, 3.5), tb, &size));  Record  tstArg = rec("Diatlov", 89, 40),  insArg = rec("Gazizov", 89, 32),  insArg0 = rec("Gazizov", 55, 3),  insArg1 = rec("Bilyk", 8, 41),  insArg2 = rec("Dudko", 80, 4.2),  insArg3 = rec("Kuznetsov", 77, 5.5);  printTb(tb, size);  printRec(selDirect(tb, 1));  printRec(insDirect(insArg, tb, 3, &size));  printTb(tb, size);  printRec(updDirect(insArg1, tb, 2, size));  printRec(delDirect(tb, 1));  printTb(tb, size);  printRec(insLinear(insArg2, tb, &size));  printTb(tb, size);  printRec(delLinear(insArg1->key, tb, size));  printTb(tb, size);  typedef struct record\* Record;  typedef struct record\*\* Table;  typedef struct key Key;  void printRec(Record rec);  void printTb(Table tb, int count);  Record rec(char\* str, unsigned short index, double value);  Record emptyRec();  struct key key(char\* str, unsigned short index);  Record selDirect(Table tb, unsigned int num);  Record insDirect(Record rec, Table tb, unsigned int num, int\* count);  Record delDirect(Table tb, unsigned int num);  Record updDirect(Record rec, Table tb, unsigned int num, int count);  int cmpKeys(Key key1, Key key2);  Record selLinear(Table tb, Key key, int count);  Record insLinear(Record rec, Table tb, int\* count);  Record delLinear(Key key, Table tb, int count);  Record updLinear(Key key, Record rec, Table tb, int count);  Table packLinear(Table tb, int\* count);  Table sort(Table tb, int count);  Record selBinary(Table tb, Key key, int count);  Record insBinary(Record rec, Table tb, int\* count);  Record delBinary(Key key, Table tb, int count);  Record updBinary(Key key, Record rec, Table tb, int count);  **table.c**  #pragma once  #include "table.h"  void printRec(Record rec) {  if (rec == 0)  printf("No such record\n");  else  if (rec->key.str == 0)  printf("EMPTY");  else  printf("key: \"%s\"; index: %d; data: %.3f%s\n", rec->key.str, rec->key.index, rec->val.data.x, rec->del == 1 ? " (is deleted)" : "");  }  void printTb(Table tb, int count) {  printf("Table:\n");  for (int i = 0; i < count; ++i) {  printRec(tb[i]);  Record selLinear(Table tb, Key key, int count) {  printf("select linear:\n");  for (int i = 0; i < count; ++i) {  if (cmpKeys(key, tb[i]->key) == 0)  return tb[i];  }  return 0;  }  Record insLinear(Record rec, Table tb, int\* count) {  printf("insert linear:\n");  for (int i = 0; i <= \*count; ++i) {  if (i == \*count || tb[i]->key.str == 0) {  tb[i] = rec;  if (i == \*count)  ++(\*count);  return tb[i];  }  }  return 0;  }  Record delLinear(Key key, Table tb, int count) {  printf("delete linear:\n");  Record res = selLinear(tb, key, count);  if (res)  res->del = 1;  return res;  }  Record updLinear(Key key, Record rec, Table tb, int count) {  printf("update linear:\n");  Record res = selLinear(tb, key, count);  if (res)  \*res = \*rec;  return res;  }  Table packLinear(Table tb, int\* count) {  printf("pack linear\n");  int newCount = 0;  for (int i = 0; i < \*count; ++i) {  if (tb[i]->del == 0)  tb[newCount++] = tb[i];  }  \*count = newCount;  return tb;  }  Table sort(Table tb, int count) {  printf("sort\n");  for (int i = 0; i < count; ++i) {  for (int k = 0; k < count - i - 1; ++k) {  if (cmpKeys(tb[k]->key, tb[k + 1]->key) > 0) {  Record t = tb[k]; | printRec(updLinear(insArg->key, insArg0, tb, size));  printTb(tb, size);  packLinear(tb, &size);  printTb(tb, size);  sort(tb, size);  printTb(tb, size);  printRec(selBinary(tb, insArg2->key, size));  printRec(insBinary(insArg3, tb, &size));  printTb(tb, size);  system("pause");  return 0;  }  **table.h**  #pragma once  #include <string.h>  #include <stdio.h>  #include <stdlib.h>  struct struct2 {  double x;  };  struct key {  char\* str;  unsigned short index;  };  struct func {  struct struct2 data;  };  struct record {  struct key key;  struct func val;  char del;  };  }  printf("\n");  }  Record rec(char\* str, unsigned short index, double value) {  Record res = (Record)malloc(sizeof(struct record));  res->key.str = str;  res->key.index = index;  res->val.data.x = value;  res->del = 0;  return res;  }  Record emptyRec() {  return rec(0, 0, 0);  }  Key key(char\* str, unsigned short index) {  Key res = { str, index };  return res;  }  Record selDirect(Table tb, unsigned int num) {  printf("select direct #%d:\n", num);  --num;  return tb[num];  }  Record insDirect(Record rec, Table tb, unsigned int num, int\* count) {  printf("insert direct #%d:\n", num);  --num;  while (\*count < num + 1)  tb[(\*count)++] = emptyRec();  tb[\*count - 1] = rec;  return tb[\*count - 1];  }  Record delDirect(Table tb, unsigned int num) {  printf("delete direct #%d:\n", num);  --num;  tb[num]->del = 1;  return tb[num];  }  Record updDirect(Record rec, Table tb, unsigned int num, int count) {  printf("update direct #%d:\n", num);  --num;  if (num > count - 1)  return 0;  tb[num] = rec;  return tb[num];  }  int cmpKeys(Key key1, Key key2) {  int cmp = strcmp(key1.str, key2.str);  if (cmp != 0)  return cmp;  return key1.index - key2.index;  }  tb[k] = tb[k + 1];  tb[k + 1] = t;  }  }  }  return tb;  }  Record selBinary(Table tb, Key key, int count) {  printf("select binary:\n");  int l = 0, r = count - 1;  do {  int m = (l + r) / 2;  int cmp = cmpKeys(tb[m]->key, key);  if (cmp == 0)  return tb[m];  if (cmp < 0)  l = m + 1;  else  r = m;  } while (l != r);  return 0;  }  Record insBinary(Record rec, Table tb, int\* count) {  printf("insert binary:\n");  Record res = selBinary(tb, rec->key, \*count);  if (!res) {  tb[(\*count)++] = rec;  res = rec;  }  return res;  }  Record delBinary(Key key, Table tb, int count) {  printf("delete binary:\n");  Record res = selBinary(tb, key, count);  if (res) {  res->del = 1;  }  return res;  }  Record updBinary(Key key, Record rec, Table tb, int count) {  printf("update binary:\n");  Record res = selBinary(tb, key, count);  if (res) {  \*res = \*rec;  }  return res;  } |