К У Р С О В А З А Д А Ч А

по учебна дисциплина:

ПИК II

Данни за студента:

char facultyId[] = “ФКСТ”;

char fullName[] = “Лъчезар Радославов Илиев”;

short fromGroup = 42;

unsigned long facNumber = 501217010;

Ръководител: Момчил Петков

Задание No 8 **дисциплина: ПИК II – курсова задача**

Студент:

Да се състави програма тип „меню” за поддръжка на информация за клиенти на банка със следните изисквания:

1. За всеки клиент на банка се съраняват следните данни:
   - уникален код (буква и цифри) – 20 символа;
   - трите имена – текст (50 символа);
   - сума - реална стойност;
   - брой месеци, през които сумата е стояла в банката – цяла стойност;

2. Информацията се съхранява в подходяща файлова структура, като се осигурят следните функции:
   а) Въвеждане на данни за нов клиент в подходящ диалогов прозорец;
   б) Търсене и извеждане на клиент по код;
   в) Изтриване на клиент с посочен код, като се запази подредеността на списъка;
   г) Извеждане на на списък на клиентите (код, имена), със сума, по-голяма от средната за всички клиенти.
3. Данните да се поддържат в динамична структура - едносвързан списък в оперативната памет на ПК.

<u>ИЗИСКВАНИЯ КЪМ ОФОРМЛЕНИЕТО</u>

Задачата да се оформи като задача, съдържаща:
- титулна страница с данни за студента, ръководителя на курсовата задача;
- текст на заданието;
- обобщен блоков алгоритъм на разработеното програмно осигуряване;
- описание на използуваните модули (функции) - прототип, входно изходни параметри и предназначение;
- общо описание за функциониране на програмата (вход/изход);
- листинг на source (изходния) код на програмата;
- резултати от изпълнението на програмата (контролен пример);
- <u>проектът да се реализира в програмната среда като проект с разделна компилация</u>.

Дата на задаване: **Преподавател:**

/ ……/

Описание на използваните функции:

var ***createStack**(double *averageBalance);
Създаване на стеков списък (менажиращ метод LIFO). Четенето става от бинарен файл, като при всяко прочитане с fread() добавя елемент към списъка, посредством функция **newItem**(); Четенето спира, когато е прочетен последният елемент от файла или при възникването на грешка. Връща като резултат указател към последният добавен елемент.

void **addNewClient**();
Отваря бинарния файл в режим “append”, потребителят въвежда данните за нов клиент и ако записът е валиден операцията е успешна и файлът се затваря, ако не – резултатът е изход от програмата.

void **writeToBin**(var *head);
Отваря бинарния файл в режим “wb” и записва всички елементи от списъка. Ако записът е валиден операцията е успешна и файлът се затваря, ако не – резултатът е изход от програмата.

void **newItem**(var **currElement, var *data);
Приема като параметри конкретния елемент, подаден от функцията ***createStack**(); и структурата с данните за конкретния елемент, прочетени от файла отново във функцията ***createStack**();. Заделя динамично памет с функцията malloc();, прави проверка дали заделянето на памет е успешно и ако е така записва данните от структурата data в полетата на конкретния елемент от списъка. Следващият елемент става равен на NULL.

void **printData**(var *head);
Приема като параметър началото на списъка. Обхожда целият сисък като отпечатва на екрана данните на всеки клиент/елемент от списъка.

void **printByCode**(var *head, char *passedCode);
Приема като параметри началото на списъка и подаденият от потребителят код на клиент. Обхожда целият списък и при окриването на съвпадение със !strcmp() отпечатва на екрана данните на избраният клиент.

var ***deleteByCode**(var *head, char *passedCode);
Приема като параметри началото на списъка и подаденият от потребителят код на клиент. Обхожда целият списък до намирането на избраният клиент със !strcmp(), премахва го от списъка с функция free(), като свързва предходният елемент със следващият елемент, с цел запазване на целостта на списъка (връзките между указателите).

void **printByBalance**(var *head, double averageBalance);
Приема като параметри началото на списъка и средната сума за всички клиенти. Обхожда списъка като оптечатва на екрана данните на всички клиенти със сума по-голяма от средната сума за всички клиенти.

void **freeMemory**(var *head);
Приема като параметър началото на списъка и освобождава заделената за него с malloc() памет.

Листинг на source (изходния) код:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct var
{
   char code[20];
   char name[50];
   double balance;
   int months;
   struct var *next;
};
typedef struct var var;

var *createStack(double *averageBalance);
void addNewClient();
void writeToBin(var *head);
void newItem(var **currElement, var *data);
void printData(var *head);
void printByCode(var *head, char *passedCode);
var *deleteByCode(var *head, char *passedCode);
void printByBalance(var *head, double averageBalance);
void freeMemory(var *head);

int main()
{
   int choice, k;
   char passedCode[20];
   double averageBalance = 0;
   var *head = NULL;

        do {
                printf("\t\t\tMenu:\n\n");
                printf("\t|1| Enter data for new client \n");
                printf("\t|2| Find client by code \n");
                printf("\t|3| Delete client by code\n");
                printf("\t|4| Show clients with balance > averageBalance \n");
                printf("\t|5| Exit\n\n");
                printf("Choose [1-5]: ");
                do {
                        k = scanf("%d", &choice);
                        while (getchar() != '\n');
                        if (choice < 1 || choice > 5)
                                printf("\tError! Try again:\n");
                } while (k != 1);
                system("CLS");
                switch (choice)
                {
                case 1:
                        printf("|1| Enter data for new client:\n");
                        addNewClient();
                        break;
```

```
                case 2:
                        printf("|2| Find client by code: ");
                        head = createStack(&averageBalance);
                        fgets(passedCode, 20, stdin);
                         if ((strlen(passedCode) > 0) && (passedCode[strlen (passedCode) - 1] == '\n'))
                                passedCode[strlen (passedCode) - 1] = '\0';
                        printByCode(head, passedCode);
                        break;
                case 3:
                        printf("|3| Delete client by code: ");
                        head = createStack(&averageBalance);
                        fgets(passedCode, 20, stdin);
                        if ((strlen(passedCode) > 0) && (passedCode[strlen (passedCode) - 1] == '\n'))
                                passedCode[strlen (passedCode) - 1] = '\0';
                        printf("\n\tBefore delete:\n");
                        printData(head);
                        head = deleteByCode(head, passedCode);
                        printf("\n\tAfter delete:\n");
                        printData(head);
                        writeToBin(head);
                        break;
                case 4:
                        averageBalance = 0;
                        head = createStack(&averageBalance);
                        printf("|4| Show clients with balance > averageBalance[%g]:\n", averageBalance);
                        printByBalance(head, averageBalance);
                        break;
                }
        } while (choice != 5);
  freeMemory(head);
  head = NULL;
  return 0;
}


var *createStack(double *averageBalance)
{
  var *head = NULL;
  var *currentItem = NULL;
  var data;
  int count = 0;
  double sum = 0;

  FILE *fp;
  if((fp = fopen("clients.bin", "rb")) == NULL)
  {
    printf("Error! File not found/does not exist...\n");
    exit(1);
  }
  while(fread(&data, sizeof(var), 1, fp))
  {
    if(ferror(fp))
      exit(1);
    sum += data.balance;
    count++;
    newItem(&currentItem, &data);
```

```
      currentItem->next = head;
      head = currentItem;
   }
   if(count > 0)
      *averageBalance += (sum/count);
   fclose(fp);
   return head;
}


void newItem(var **currElement, var *data)
{
   *currElement = (var *)malloc(sizeof(var));
   if(*currElement == NULL)
   {
      printf("Error! Exiting program...\n");
      exit(1);
   }
   strcpy((*currElement)->code, data->code);
   strcpy((*currElement)->name, data->name);
   (*currElement)->balance = data->balance;
   (*currElement)->months = data->months;

   (*currElement)->next = NULL;
}


void addNewClient()
{
   var client;
   FILE *fp;
   if((fp=fopen("clients.bin", "ab"))==NULL)
   {
      printf("Error! Can not open/create file for write...\n");
      exit(1);
   }
   printf("Enter code: ");
   fgets(client.code, 20, stdin);
   if ((strlen(client.code) > 0) && (client.code[strlen(client.code) - 1] == '\n'))
      client.code[strlen(client.code) - 1] = '\0';
   printf("Enter name: ");
   fgets(client.name, 50, stdin);
   if ((strlen(client.name) > 0) && (client.name[strlen(client.name) - 1] == '\n'))
      client.name[strlen(client.name) - 1] = '\0';
   printf("Enter balance: ");
   scanf("%lf", &client.balance);
   printf("Enter months: ");
   scanf("%d", &client.months);
   if(fwrite(&client, sizeof(var), 1, fp)!=1)
   {
      printf("Error writing new client!\nExiting...");
      exit(1);
   }
   if(ferror(fp))
      exit(1);
   fclose(fp);
}
```

```
void writeToBin(var *head)
{
   FILE *fp;
   if((fp=fopen("clients.bin", "wb"))==NULL)
   {
      printf("Error! Can not open/create file for write...\n");
      exit(1);
   }
   var *currElement = NULL;
   for(currElement = head; currElement != NULL; currElement = currElement->next)
   {
      if(fwrite(&(*currElement), sizeof(var), 1, fp)!=1)
      {
         printf("Error writing new client!\nExiting...");
         exit(1);
      }
      if(ferror(fp))
         exit(1);
   }
   fclose(fp);
}


void printData(var *head)
{
   var *currElement = NULL;
   for(currElement = head; currElement != NULL; currElement = currElement->next)
   {
      printf("\nCode: %s\n", currElement->code);
      printf("Name: %s\n", currElement->name);
      printf("Balance: %g\n", currElement->balance);
      printf("Months: %d\n", currElement->months);
   }
}


void printByCode(var *head, char *passedCode)
{
   var *currElement = NULL;
   for(currElement = head; currElement != NULL; currElement = currElement->next)
   {
      if(!strcmp((currElement->code), passedCode))
      {
         printf("\nCode: %s\n", currElement->code);
         printf("Name: %s\n", currElement->name);
         printf("Balance: %g\n", currElement->balance);
         printf("Months: %d\n", currElement->months);
      }
   }
}
```

```
var *deleteByCode(var *head, char *passedCode)
{
   var *prevElement=head;
   var *currElement=head;
   int flag = 0;
   while(currElement!=NULL)
   {
      if(!strcmp((currElement->code), passedCode))
      {
         if(currElement == head)
         {
            head = head->next;
            prevElement = head;
            flag = 1;
         }
         else
            prevElement->next = currElement->next;
         free(currElement);
         currElement = prevElement;
      }
      prevElement = currElement;
      if(flag == 0)
         currElement = currElement->next;
      flag = 0;
   }
   return head;
}


void printByBalance(var *head, double averageBalance)
{
   var *currElement = NULL;
   for(currElement = head; currElement != NULL; currElement = currElement->next)
   {
      if(currElement->balance > averageBalance)
      {
         printf("\nCode: %s\n", currElement->code);
         printf("Name: %s\n", currElement->name);
         printf("Balance: %g\n", currElement->balance);
         printf("Months: %d\n", currElement->months);
      }
   }
}


void freeMemory(var *head)
{
   var *currElement = head;
   while(head != NULL)
   {
      head = head->next;
      free(currElement);
      currElement = head;
   }
}
```

Резултати от изпълнението на програмата (контролен пример):