# Обобщен блоков алгоритъм:

# Описание на използваните модули (функции):

***trip.h* – Функции нужни за обработката на структурата Trip**

**int menu() –** извеждане на опциите за обработка на структурата

**return –** *choice*, с който се указва коя опция да се изпълни

**struct node\* addTrip(struct node \*head) –** добавяне на екскурзия в списъка

**head –** указател към началото на структурата

**return –** *head* , указател към началото на създадения списък

**Trip getData (struct node \*head) –** въвеждане на информация за екскурзия от потребителя

**head –** указател към началото на структурата

**return –** *data,* въведената информация за екскурзията

**int unique (struct node \*head, char \*str) –** проверява дали въведения уникален код вече съществува в списъка

**head –** указател към началото на структурата

**str –** указател към въведения уникален код

**return –** *0*, ако уникалният код вече съществува в списъка или *1*, ако го няма в него

**int correctDate (int day, int month, int year) –** проверява дали въведената от потребителя дата е валидна

**day –** въведеният ден от потребителя

**month –** въведеният месец от потребителя

**year –** въведената година от потребителя

**return –** *0*, ако датата е невалидна или *1*, ако е

**void printList (struct node \*head) –** извежда на екрана списъка с всички екскурзии

**head –** указател към началото на структурата

**void printLongestTrip (struct node \*head) -**  извежда на екрана най-дългата екскурзия

**head –** указател към началото на структурата

**void printExpiredTrips (struct node \*head) –** извежда на екрана всички екскурзии с изтекла дата на заминаване

**head –** указател към началото на структурата

**struct node\* deleteItem(struct node \*head) –** изтриване на данните за екскурзия с посочен уникален код

**head –** указател към началото на структурата

**return –** *head* , указател към началото на създадения списък

**void saveListToFile(struct node \*head) –** записване на списъка в двоичен файл

**head –** указател към началото на структурата

**struct node \*readFile(struct node \*\*head) –** прочитане на двоичен файл и добавяне на информацията му към списъка

**head –** указател към началото на структурата

**return –** *head,* указател към началото на прочетения списък

# Общо описание за функциониране на програмата (вход/изход)

Когато се стартира програмата, се показва менюто с опциите и на потребителя се предоставя възможност да избере желаното. В менюто се включват следните обработки: добавяне и изтриване на екскурзия, отпечатване на най-дългата и на всички изтекли екскурзии, както и извеждане на целия списък и прочитане на двоичен файл. Когато се въвеждат екскурзии се проверява всяка информация, която се въведе (проверка за дублиране на уникалния код, валидност на датата и т.н.). При извеждане на екскурзиите с изтекъл срок на заминаване, потребителят указва датата, проверяват се датите на всички екскурзии от списъка и отпечтва само тези, които са изтекли, а когато се изисква да се изведе най-дългата екскурзия се обхождат всички и се сравняват продължителностите им. При възможността за изтриване на екскурзия се въвежда уникален код проверява се целият списък за съвпадение с въведеното и ако има такова екскурзията се изтрива.

# Листинг на изходния код на програмата:

**header.h**

/\*

\* Including libararies and defining global variables

\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#include<math.h>

#define MAX 50

char fileName[MAX];

typedef struct{

 char id[MAX];

 int day;

 int month;

 int year;

 int duration;

 float price;

}Trip;

struct node{

 Trip data;

 struct node \*next;

};

/\*

\* Showing menu's options

\*

\* @return entered choice

\*/

int menu();

/\*

\* Adding trip to list

\*

\* @return the head of the list

\*/

struct node\* addTrip(struct node \*head);

/\*

\* Entering data for a trip

\*

\* @return entered data

\*/

Trip getData (struct node \*head);

/\*

\* Checking for unique ID

\*

\* @return 0 if ID already exist or 1 if it is not in the list

\*/

int unique (struct node \*head, char \*str);

/\*

\* Checking date

\*

\* @return 0 if date is incorrect or 1 if it is correct

\*/

int correctDate (int day, int month, int year);

/\*

\* Printing data for all trips

\*

\* @return none

\*/

void printList (struct node \*head);

/\*

\* Printing data for longest trip

\*

\* @return none

\*/

void printLongestTrip (struct node \*head);

/\*

\* Printing data for expired trips

\*

\* @return none

\*/

void printExpiredTrips (struct node \*head);

/\*

\* Deleting trip from list

\*

\* @return the head of the list

\*/

struct node\* deleteItem(struct node \*head);

/\*

\* Saving the list to file

\*

\* @return none

\*/

void saveListToFile(struct node \*head);

/\*

\* Reading data from file

\*

\* @return head pointer

\*/

struct node \*readFile(struct node \*\*head);

**main.c**

#include"header.h"

int main(){

 int choice;

 struct node \*head;

 head=NULL;

 do{

 system("cls");

 choice=menu();

 switch(choice){

 case 1:{

 head=addTrip(head);

 saveListToFile(head);

 break;

 }

 case 2:{

 printList(head);

 break;

 }

 case 3:{

 printLongestTrip(head);

 break;

 }

 case 4:{

 printExpiredTrips(head);

 break;

 }

 case 5:{

 head=deleteItem(head);

 break;

 }

 case 6:{

 head=readFile(&head);

 break;

 }

 }

 system("PAUSE");

 }while(choice!=0);

 return 0;

}

# trip.c

# #include "header.h"

# int menu()

# {

#  int choice;

#  fflush(stdin);

#  printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

#  printf("| 1.Add a trip. |\n");

#  printf("| 2.Print list. |\n");

#  printf("| 3.Print Longest trip. |\n");

#  printf("| 4.Print expired trips. |\n");

#  printf("| 5.Delete an item. |\n");

#  printf("| 6.Read the list from file. |\n");

#  printf("| 0.Exit |");

#  printf("\n|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

#  printf("\nChoice: ");

#  scanf("%d",&choice);

#  printf("\n");

#  return choice;

# }

# struct node\* addTrip(struct node \*head){

#  struct node \*newNode;

#  newNode=(struct node \*)malloc(sizeof(struct node));

#  if(!newNode){

#  printf("\n-->Cannot allocate memory!<--\n\n");

#  exit(1);

#  }

#  newNode->data=getData(head);

#  newNode->next=head;

#  head=newNode;

#  return head;

# }

# Trip getData(struct node \*head){

#  Trip data;

#  fflush(stdin);

#  do{

#  printf("\nEnter ID : ");

#  scanf("%s",data.id);

#  }while(!unique(head,data.id));

#  fflush(stdin);

#  do{

#  printf("\nEnter day : ");

#  scanf("%d",&data.day);

#  printf("\nEnter month : ");

#  scanf("%d",&data.month);

#  printf("\nEnter year : ");

#  scanf("%d",&data.year);

#  }while(!correctDate(data.day,data.month,data.year));

#  fflush(stdin);

#  do{

#  printf("\nEnter duration : ");

#  scanf("%d",&data.duration);

#  if (data.duration<1){

#  printf("\n-->Wrong duration<--\n");

#  }

#  }while(data.duration<1);

#  fflush(stdin);

#  do{

#  printf("\nEnter price : ");

#  scanf("%f",&data.price);

#  if (data.price<1){

#  printf("\n-->Wrong price<--\n");

#  }

#  }while(data.price<1);

#  return data;

# }

# int unique(struct node \*head,char \*str){

#  struct node \*current=NULL;

#  current=head;

#  while(current){

#  if(strcmp(current->data.id,str)==0){

#  printf("\n-->The ID already exist in the list!<--\n");

#  return 0;

#  }

#  current=current->next;

#  }

#  return 1;

# }

# int correctDate(int day, int month, int year){

#  if (year<1){

#  printf("\n-->Wrong year!<--\n");

#  return 0;

#  }

#  else{

#  if (year%4==0){

#  if(month<1||month>12){

#  printf("\n-->Wrong month!<--\n");

#  return 0;

#  }

#  else if (month==1||month==3||month==5||month==7||month==8||month==10||month==12){

#  if (day>31||day<1){

#  printf("\n-->Wrong day!<--\n");

#  return 0;

#  }

#  else {

#  return 1;

#  }

#  }

#  else if(month==2){

#  if(day>29||day<1){

#  printf("\n-->Wrong day!<--\n");

#  return 0;

#  }

#  else{

#  return 1;

#  }

#  }

#  else{

#  if (day>30||day<1){

#  printf("\n-->Wrong day!<--\n");

#  return 0;

#  }

#  else{

#  return 1;

#  }

#  }

#  }

#  else{

#  if(month>12||month<1){

#  printf("\n-->Wrong month!<--\n");

#  return 0;

#  }

#  else if (month==1||month==3||month==5||month==7||month==8||month==10||month==12){

#  if (day<1||day>31){

#  printf("\n-->Wrong day!<--\n");

#  return 0;

#  }

#  else {

#  return 1;

#  }

#  }

#  else if(month==2){

#  if(day<1||day>28){

#  printf("\n-->Wrong day!<--\n");

#  return 0;

#  }

#  else{

#  return 1;

#  }

#  }

#  else {

#  if (day<1||day>30){

#  printf("\n-->Wrong day!<--\n");

#  return 0;

#  }

#  else{

#  return 1;

#  }

#  }

#  }

#  }

# }

# void printList (struct node \*head){

#  struct node \*current;

#  current=head;

#  while(1){

#  if(!current){

#  printf("\n-->The list is emtpy!<--\n\n");

#  break;

#  }

#  printf ("\nList data: \n");

#  printf ("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

#  while(current){

#  printf ("\nID : %s",current->data.id);

#  printf ("\nDate : %02d.%02d.%04d",current->data.day,current->data.month,current->data.year);

#  printf ("\nDuration : %i",current->data.duration);

#  printf ("\nPrice : %.2f $",current->data.price);

#  current=current->next;

#  }

#  printf ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

#  break;

#  }

#

# }

# void printLongestTrip (struct node \*head){

#  struct node \*current, \*longestTrip;

#  int max=0;

#  current=head;

#  while(1){

#  if(!current){

#  printf("\n-->The list is emtpy!<--\n\n");

#  break;

#  }

#  while (current){

#  if(current->data.duration>max){

#  max=current->data.duration;

#  longestTrip=current;

#  }

#  current=current->next;

#  }

#  printf ("\nThe longest trip data: ");

#  printf ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

#  printf ("\nID : %s",longestTrip->data.id);

#  printf ("\nDate : %02d.%02d.%04d", longestTrip->data.day,longestTrip->data.month,longestTrip->data.year);

#  printf ("\nDuration : %i",longestTrip->data.duration);

#  printf ("\nPrice : %.2f $",longestTrip->data.price);

#  printf ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

#  break;

#  }

# }

# void printExpiredTrips (struct node \*head){

#  struct node \*current;

#  int day, month, year, f;

#  f=0;

#  do{

#  printf("\nEnter day : ");

#  scanf("%i",&day);

#  printf("\nEnter month : ");

#  scanf("%i",&month);

#  printf("\nEnter year : ");

#  scanf("%i",&year);

#  }while(!correctDate(day,month,year));

#  current=head;

#  while(1){

#  if(!current){

#  printf("\n-->The list is emtpy!<--\n\n");

#  break;

#  }

#  printf ("\nThe expired trips: ");

#  printf ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

#  while (current){

#  if(current->data.year<=year){

#  if(current->data.month<=month){

#  if (current->data.day<=day){

#  printf ("\nID : %s",current->data.id);

#  printf ("\nDate : %02d.%02d.%04d",current->data.day,current->data.month,current->data.year);

#  printf ("\nDuration : %i",current->data.duration);

#  printf ("\nPrice : %.2f $",current->data.price);

#  f=1;

#  }

#  }

#  }

#  current=current->next;

#  }

#  if(f==0){

#  printf("\n-->There are not expired trips!<--");

#  }

#  printf ("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n");

#  break;

#  }

# }

# struct node \*deleteItem(struct node \*head){

#  char word[MAX];

#  struct node \*previous=NULL;

#  struct node \*current=head;

#  int f=0;

#  fflush(stdin);

#  if(head==NULL){

#  printf("\n-->The list is empty!<--\n\n");

#  return 0;

#  }

#  printf("\nEnter ID : ");

#  scanf("%s",word);

#  for(current=head;current!=NULL;previous=current,current=current->next){

#  if(strcmp(current->data.id,word)==0){

#  f=1;

#  if(previous==NULL){

#  head=current->next;

#  }

#  else{

#  previous->next=current->next;

#  }

#  free(current);

#  }

#  }

#  if(f==1){

#  printf("\n-->The item was deleted!<--\n\n");

#  }

#  else {

#  printf("\n-->The item is not in the list!<--\n\n");

#  }

#  return head;

# }

# void saveListToFile(struct node \*head){

#  FILE \*fp;

#  struct node \*current;

#  printf("\nEnter a name of the file : ");

#  scanf("%s",fileName);

#  current=head;

#  fp=fopen(fileName,"wb");

#  while (1){

#  if(head==NULL){

#  printf("\n-->The list is empty!<--\n");

#  break;

#  }

#  if(fp==NULL){

#  printf("\n-->The file doesn't exist!<--\n");

#  break;

#  }

#  while(current!=NULL){

#  fwrite(&current->data,sizeof(current->data),1,fp);

#  current=current->next;

#  }

#  printf("\n-->The data was saved!<--\n\n");

#  break;

#  }

#  fclose(fp);

# }

# struct node \*readFile(struct node \*\*head) {

#  FILE \*fp;

#  char fName[MAX];

#  struct node \*current;

#  printf("\nEnter a name of the file : ");

#  scanf("%s",fName);

#  fp=fopen(fName,"rb");

#  if(fp==NULL){

#  printf("\n-->Can not find a file!<--\n\n");

#  }

#  else{

#  do{

#  current=(struct node\*)malloc(sizeof(struct node));

#  current->next=NULL;

#  if(!(fread(&(current->data), sizeof(current->data),1,fp))){

#  break;

#  }

#  current->next=\*head;

#  \*head=current;

#  }while(1);

#  printf("\n-->The data was read!<--\n\n");

#  }

#  fclose(fp);

#  return \*head;

# }

# Контролни примери

*Въвеждане на екскурзия:*



*Показване на целия списък:*



*Показване на най-дългата екскурзия:*



*Извеждане на изтеклите екскурзии:*



*Изтриване на елемент от списъка:*



*Прочитане на файл:*

