**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 8**

**по дисциплине «Программирование»**

Тема: Линейные односвязные списки.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3312 |  | Шарапов И. Д. |
| Преподаватель |  | Аббас С. А. |

Санкт-Петербург

2024

**Содержание**

[Цель работы 3](#_Toc162175841)

[Задание (Вариант 1) 3](#_Toc162175842)

[Постановка задачи и описание решения 3](#_Toc162175843)

[Описание переменных 4](#_Toc162175844)

[Схема подалгоритма 5](#_Toc162175845)

[Текст программы 5](#_Toc162175846)

[Контрольные примеры 10](#_Toc162175847)

[Содержимое файлов 13](#_Toc162175848)

[Примеры выполнения программы 15](#_Toc162175849)

[Выводы 18](#_Toc162175850)

# Цель работы

Целью работы является изучение линейных односвязных списков в языке Си, а также работа с указателями на структуры.

# Задание (Вариант 1)

Разработать подалгоритм и написать функцию, вставляющую в односвязный список получаемые данные перед заданным по номеру элементом. Номер элемента задаётся с конца списка. При недостаточном количестве элементов в списке данные вставить в начало списка.

# Постановка задачи и описание решения

Для выполнения задания в код прошлой лабораторной работы было добавлено 4 функции.

Функция *make\_list()* инициализирует список: выделяет память для головы и заполняет её поля.

Функция *create\_node()* выделяет память для нового элемента списка и заполняет его с помощью функции *fill\_struct()*.

Функция *select\_by\_order()* перебирает все элементы списка до тех пор, пока не найдёт элемент с указанным индексом и возвращает указатель на его. Если элемента с таким индексом не существует, функция выведет соответствующее сообщение и вернёт указатель на первый элемент списка.

Функция *add()* вставляет в список элемент. Для этого строка преобразуется в элемент списка с помощью функции *create\_node()* и ищется элемент, перед которым нужно вставить, с помощью функции *select\_by\_order().* В зависимости от положения элемента меняются указатели соседних элементов, тем самым новый элемент добавляется в список.

# Описание переменных

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя переменной | Тип | Назначение |
| Функция *ListOfAthlete \*make\_list()* | | | |
| 1 | ph | ListOfAthlete \* | Указатель на голову списка |
| Функция *NodeOfList \*create\_node(const char \*text)* | | | |
| 1 | text | char \* | Исходная строка с данными |
| 2 | new\_node | NodeOfList \* | Новый элемент списка |
| 3 | copytext | char \* | Копия исходной строки |
| Функция *NodeOfList \*select\_by\_order(const ListOfAthlete \*list, int ind)* | | | |
| 1 | list | ListOfAthlete \* | Указатель на голову списка |
| 2 | ind | int | Номер элемента, который нужно найти |
| 3 | q | NodeOfList \* | Временный элемент для перебора |
| 4 | n | int | Длина списка |
| Функция *void add(ListOfAthlete \*list, NodeOfList \*new\_node, int ind)* | | | |
| 1 | list | ListOfAthlete \* | Указатель на голову списка |
| 2 | new\_node | NodeOfList \* | Элемент, который нужно вставить |
| 3 | ind | int | Номер элемента, перед которым нужно вставить |
| 4 | cur\_node | NodeOfList \* | Указатель на элемент, перед которым нужно вставить |
| 5 | q | NodeOfList \* | Временный элемент для перебора |

# Схема подалгоритма



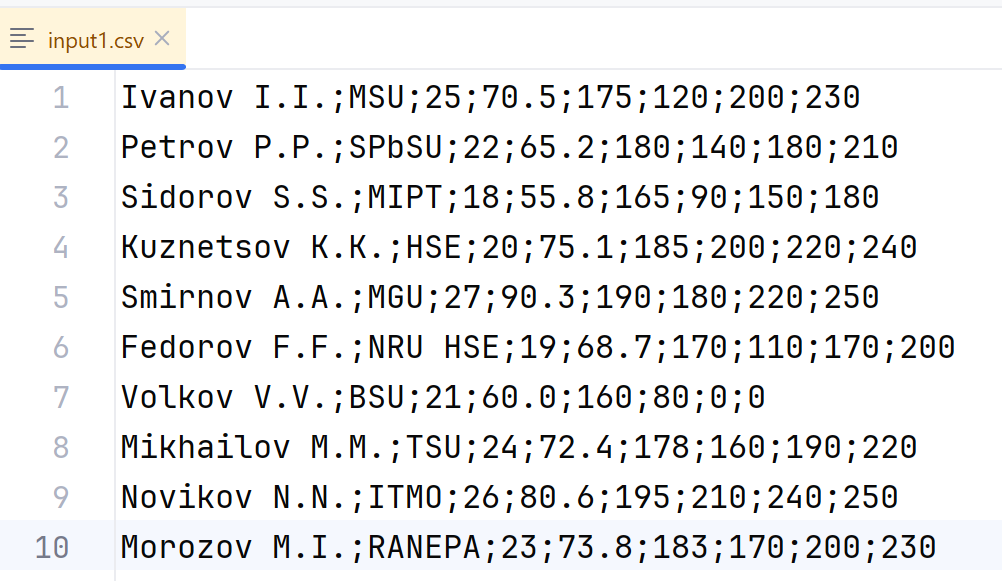
# Текст программы

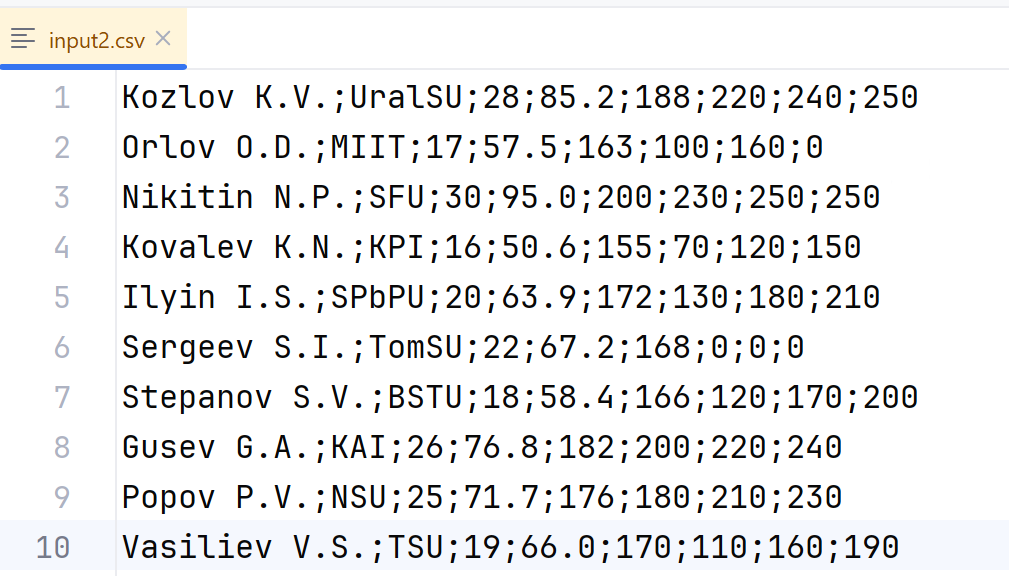
|  |
| --- |
| **#include <stdio.h> #include <stdlib.h> #include <string.h>  typedef struct Athlete {  char \*name;  char \*university;  int age;  float weight;  int height;  int result[3];  float index; } Athlete;  typedef struct NodeOfList {  Athlete \*data;  struct NodeOfList \*next; } NodeOfList;  typedef struct ListOfAthlete {  int length;  NodeOfList \*first; } ListOfAthlete;   int from\_str\_to\_int(char \*x) {  int ans = 0;   while (\*x != '\0') {  ans = ans \* 10 + (\*x - '0');  ++x;  }  return ans; }  float from\_str\_to\_float(char \*x) {  float ans = 0, a = 10, b = 1;   while (\*x != '\0') {  if (\*x == '.' || \*x == ',') {  a = 1;  b = 10;  } else {  ans = ans \* a + (float) (\*x - '0') / b;  if (b > 1) b \*= 10;  }  ++x;  }  return ans; }  void from\_str\_to\_int\_mas(char \*x, int \*mas) {  int ind = 0, j = 0;   while (x[j] != '\0') {  if (x[j] == ';') {  x[j] = '\0';  mas[ind++] = from\_str\_to\_int(x);  x += j + 1;  j = -1;  }  ++j;  }  mas[ind] = from\_str\_to\_int(x); }  Athlete \*fill\_struct(char \*str) {  Athlete \*user = NULL;  char \*word = str;  int ind = 0, tt;  char \*pole[5];   user = (Athlete \*) malloc(sizeof(Athlete));  if (user != NULL) {  for (tt = 0; str[tt] != '\n' && str[tt] != '\0'; ++tt) {  if (str[tt] == ';' && ind < 5) {  str[tt] = '\0';  pole[ind++] = word;  word = str + tt + 1;  }  }  str[tt] = '\0';  user->name = pole[0];  user->university = pole[1];  user->age = from\_str\_to\_int(pole[2]);  user->weight = from\_str\_to\_float(pole[3]);  user->height = from\_str\_to\_int(pole[4]);  from\_str\_to\_int\_mas(word, user->result);  user->index = (float) (user->result[0] + user->result[1] + user->result[2]) / user->weight;  }  return user; }  ListOfAthlete \*make\_list() {  ListOfAthlete \*ph = NULL;  ph = (ListOfAthlete \*) malloc(sizeof(ListOfAthlete));  if (ph != NULL) {  ph->length = 0;  ph->first = NULL;  }  return ph; }  NodeOfList \*create\_node(const char \*text) {  NodeOfList \*new\_node = NULL;  char \*copytext = NULL;   new\_node = (NodeOfList \*) malloc(sizeof(NodeOfList));  copytext = (char \*) malloc((strlen(text) + 1) \* sizeof(char));  if (new\_node && copytext) {  strcpy(copytext, text);  new\_node->data = fill\_struct(copytext);  new\_node->next = NULL;  }  return new\_node; }  NodeOfList \*select\_by\_order(const ListOfAthlete \*list, int ind) {  NodeOfList \*q = list->first;  int n = list->length;   if (ind < 0 || ind > n) {  ind = n - 1;  printf("The index is out of range!\n"  "The item will be inserted at the top of the list.\n");  }  for (int i = 1; i < n - ind; ++i) q = q->next;  return q; }  void add(ListOfAthlete \*list, NodeOfList \*new\_node, int ind) {  NodeOfList \*cur\_node = select\_by\_order(list, ind);  NodeOfList \*q = NULL;   if (list && new\_node && cur\_node) {  if (list->first == cur\_node) {  new\_node->next = cur\_node;  list->first = new\_node;  ++list->length;  } else {  q = list->first;  while (q != NULL) {  if (q->next == cur\_node) {  q->next = new\_node;  new\_node->next = cur\_node;  ++list->length;  q = NULL;  } else q = q->next;  }  }  } }  void free\_list(ListOfAthlete \*list) {  NodeOfList \*cur\_node = list->first;  NodeOfList \*next\_node;   free(list);  while (cur\_node != NULL) {  next\_node = cur\_node->next;  free(cur\_node->data);  free(cur\_node);  cur\_node = next\_node;  } }  void print\_line() {  printf("+");  for (int i = 0; i < 22; printf("-"), ++i);  printf("+------------+-----+--------+--------"  "+------+------+------+-------+\n"); }  void print\_node(Athlete \*node) {  printf("| %-20s | %-10s | %-3i | %0.1f ", node->name, node->university,  node->age, node->weight);  if (node->weight < 100) printf(" ");  printf("| %-6i | %-4i | %-4i | %-4i ", node->height, node->result[0],  node->result[1], node->result[2]);  printf("| %0.3f |\n", node->index); }  void pprint(const ListOfAthlete \*list) {  NodeOfList \*cur\_node = list->first;   print\_line();  printf("| Name | University | Age | Weight | Height "  "| Res1 | Res2 | Res3 | Index |\n");  print\_line();  while (cur\_node != NULL) {  print\_node(cur\_node->data);  cur\_node = cur\_node->next;  }  print\_line(); }  NodeOfList \*\*get\_mas(const ListOfAthlete \*list) {  NodeOfList \*cur\_node = list->first;  NodeOfList \*\*mas = NULL;   mas = (NodeOfList \*\*) malloc(list->length \* sizeof(NodeOfList \*));  if (mas != NULL) {  for (int i = 0; cur\_node != NULL; ++i) {  mas[i] = cur\_node;  cur\_node = cur\_node->next;  }  }  return mas; }  void my\_swap(NodeOfList \*\*mas, ListOfAthlete \*list, int i, int j) {  NodeOfList \*q;   if (i == 0) {  list->first = mas[j];  } else {  mas[i - 1]->next = mas[j];  }  mas[j - 1]->next = mas[i];  q = mas[j]->next;  mas[j]->next = mas[i]->next;  mas[i]->next = q;  q = mas[i];  mas[i] = mas[j];  mas[j] = q; }  void sort\_list(ListOfAthlete \*list, int param) {  NodeOfList \*\*mas = get\_mas(list);  int n = list->length;   for (int i = 0; i < n; ++i) {  for (int j = i; j < n; ++j) {  if ((param == 1 && mas[i]->data->age > mas[j]->data->age) ||  (param == 2 && mas[i]->data->weight > mas[j]->data->weight) ||  (param == 3 && mas[i]->data->height > mas[j]->data->height) ||  (param == 4 && mas[i]->data->index > mas[j]->data->index)) {  my\_swap(mas, list, i, j);  }  }  }  free(mas); }  char \*m\_strlwr(const char \*str) {  char \*new\_str = NULL;  new\_str = (char \*) malloc((strlen(str) + 1) \* sizeof(char));   if (new\_str != NULL) {  strcpy(new\_str, str);  strlwr(new\_str);  }  return new\_str; }  void sorted(int \*mas, const ListOfAthlete \*list, int param) {  NodeOfList \*cur\_node, \*min\_node;  int ind;   for (int j = 0; j < list->length; ++j) {  cur\_node = list->first;  min\_node = NULL;  for (int i = 0; cur\_node != NULL && i < list->length; ++i, cur\_node = cur\_node->next) {  if (mas[i] == 1) {  if ((min\_node == NULL) ||  ((param == 1 && min\_node->data->age > cur\_node->data->age) ||  (param == 2 && min\_node->data->weight > cur\_node->data->weight) ||  (param == 3 && min\_node->data->height > cur\_node->data->height) ||  (param == 4 && min\_node->data->index > cur\_node->data->index))) {  min\_node = cur\_node;  ind = i;  }  }  }  if (min\_node != NULL) {  mas[ind] = 2;  print\_node(min\_node->data);  }  }   for (int j = 0; j < list->length; ++j) {  if (mas[j] == 2) mas[j] = 1;  } }  void find(ListOfAthlete \*list, int param) {  NodeOfList \*cur\_node = list->first;  char x[128], \*str, \*new\_str;  int mas[list->length], fl = 0, ch;   printf("Enter the search string:\n");  getchar();  fgets(x, sizeof(x), stdin);  x[strlen(x) - 1] = '\0';  strlwr(x);  for (int i = 0; cur\_node != NULL && i < list->length; ++i) {  if (param == 1) str = cur\_node->data->name;  else str = cur\_node->data->university;  new\_str = m\_strlwr(str);  if (strstr(new\_str, x) != NULL) {  if (fl == 0) {  print\_line();  printf("| Name | University | Age | Weight "  "| Height | Res1 | Res2 | Res3 | Index |\n");  print\_line();  }  print\_node(cur\_node->data);  fl = 1;  mas[i] = 1;  } else {  mas[i] = 0;  }  free(new\_str);  cur\_node = cur\_node->next;  }  if (fl == 0) {  printf("No matches found!\n");  } else {  print\_line();  do {  printf("Select a field to sort by or exit:\n"  "1 = age\n"  "2 = weight\n"  "3 = height\n"  "4 = index\n"  "0 = exit\n"  "Enter only one number!\n");  scanf("%i", &ch);  if (ch < 0 || 4 < ch) {  printf("Invalid command!\n");  } else if (ch != 0) {  print\_line();  printf("| Name | University | Age | Weight "  "| Height | Res1 | Res2 | Res3 | Index |\n");  print\_line();  sorted(mas, list, ch);  print\_line();  } else {  printf("To display the data, enter the command \"!print\"\n"  "To find athletes, enter the command \"!find\"\n"  "To sort the data, enter the command \"!sort\"\n"  "To add new data, enter the command \"!add\"\n"  "To end the program, enter the command \"!end\"\n");  }  } while (ch != 0);  } }  int main() {  ListOfAthlete \*list;  NodeOfList \*cur\_node = NULL, \*last\_node = NULL;  char filename[128], text[1024], str[128];  FILE \*f;  int ch;   printf("Please enter the file name:\n");  scanf("%s", filename);  f = fopen(filename, "r");  while (f == NULL) {  printf("Something went wrong!\n"  "Perhaps such a file does not exist.\n"  "Please enter the file name again:\n");  scanf("%s", filename);  f = fopen(filename, "r");  }   list = make\_list();  while (fgets(text, sizeof(text), f)) {  if (list->length == 0) {  cur\_node = create\_node(text);  list->first = cur\_node;  last\_node = cur\_node;  } else {  cur\_node = create\_node(text);  last\_node->next = cur\_node;  last\_node = cur\_node;  }  ++list->length;  }  printf("The file has successfully been processed!\n");  fclose(f);   printf("To display the data, enter the command \"!print\"\n"  "To find athletes, enter the command \"!find\"\n"  "To sort the data, enter the command \"!sort\"\n"  "To add new data, enter the command \"!add\"\n"  "To end the program, enter the command \"!end\"\n");   do {  scanf("%s", str);  if (!strcmp(str, "!end")) {  printf("Goodbye!\n");  } else if (!strcmp(str, "!print")) {  pprint(list);  } else if (!strcmp(str, "!find")) {  printf("Select a field to find by:\n"  "1 = name\n"  "2 = university\n"  "Enter only one number!\n");  scanf("%i", &ch);  if (ch < 1 || 2 < ch) {  printf("Invalid command!\n");  } else {  find(list, ch);  }  } else if (!strcmp(str, "!sort")) {  printf("Select a field to sort by:\n"  "1 = age\n"  "2 = weight\n"  "3 = height\n"  "4 = index\n"  "Enter only one number!\n");  scanf("%i", &ch);  if (ch < 1 || 4 < ch) {  printf("Invalid command!\n");  } else {  sort\_list(list, ch);  printf("The data has been successfully sorted!\n");  pprint(list);  }  } else if (!strcmp(str, "!add")) {  printf("Enter data of the athlete in format:\n"  "name;university;age;weight;height;result1,result2,result3\n");  getchar();  fgets(text, sizeof(text), stdin);  printf("Enter the number of the item indicated at the end\n"  "of the list before which you want to insert the athlete:\n");  scanf("%i", &ch);  --ch;  add(list, create\_node(text), ch);  printf("The item has been successfully inserted!\n");  } else {  printf("Unknown command!\n");  }  } while (strcmp(str, "!end") != 0);   free\_list(list);  return 0; }** |

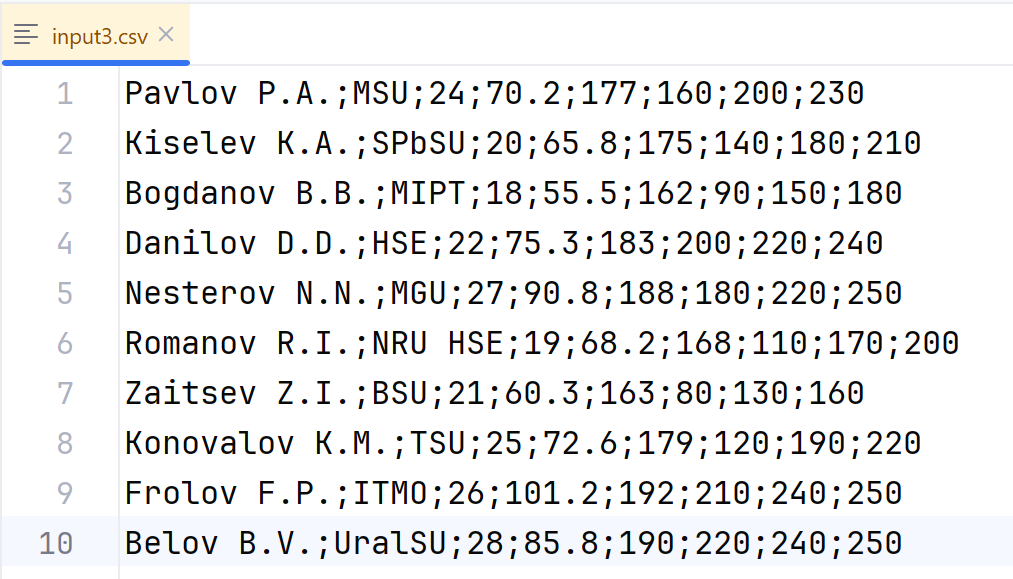
# Контрольные примеры

|  |  |  |
| --- | --- | --- |
| № | Исходные данные | Результаты |
| 1 | input1.csv  !print  !add  Dmitriev D.S.;TomSU;23;67.8;172;150;190;220  3  !print  !end |  |
| 2 | input2.txt  input2.csv  !add  Sorokin S.O.;MIIT;17;57.9;165;100;160;190  15  !print  !end |  |
| 3 | input3.csv  !add  Sokolov S.V.;TSU;20;66.6;171;110;160;190  4  !print  !add  Vorobyov V.I.;MSU;28;86.2;186;220;240;250  7  !print  !end |  |

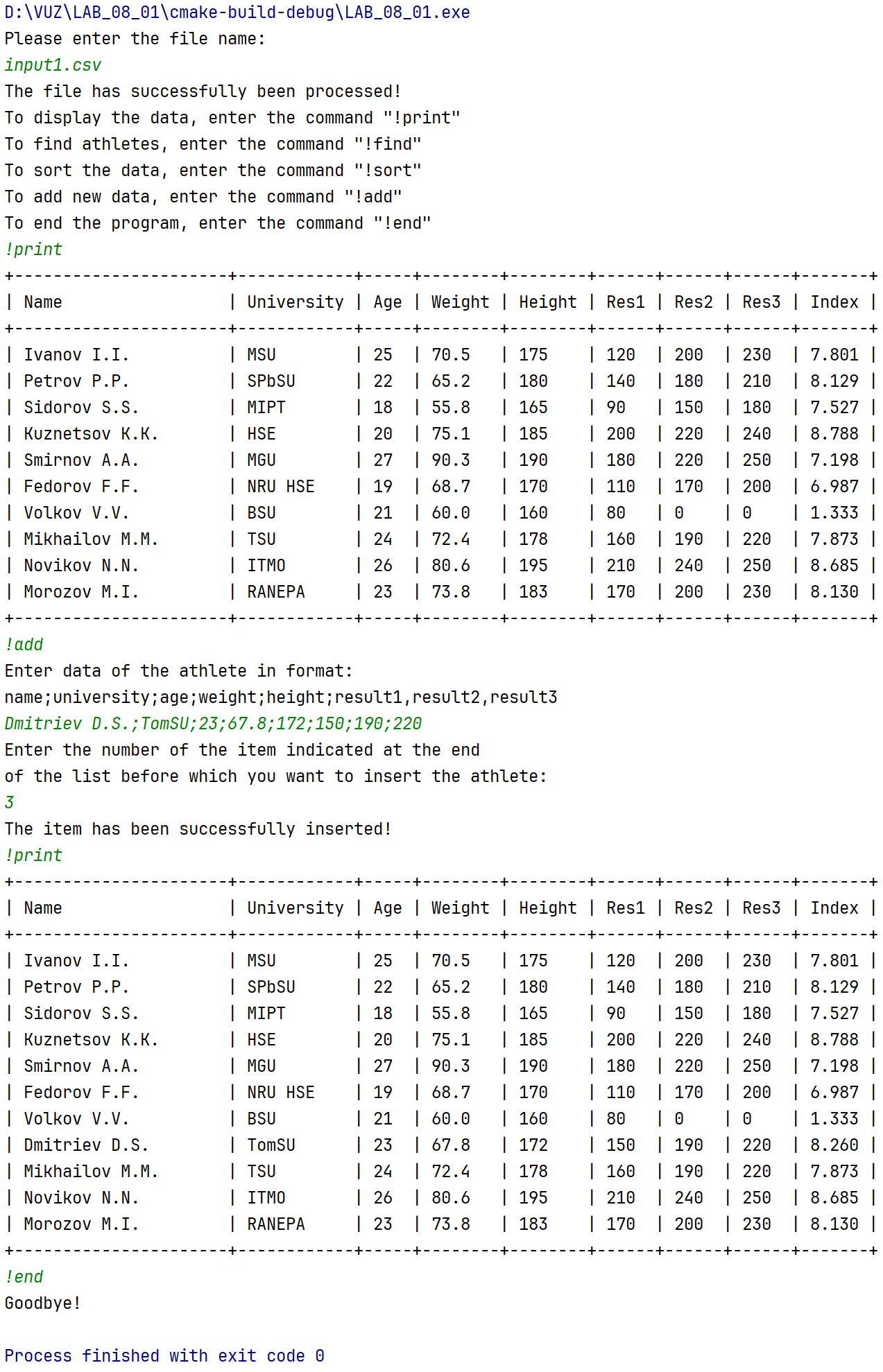
# Содержимое файлов

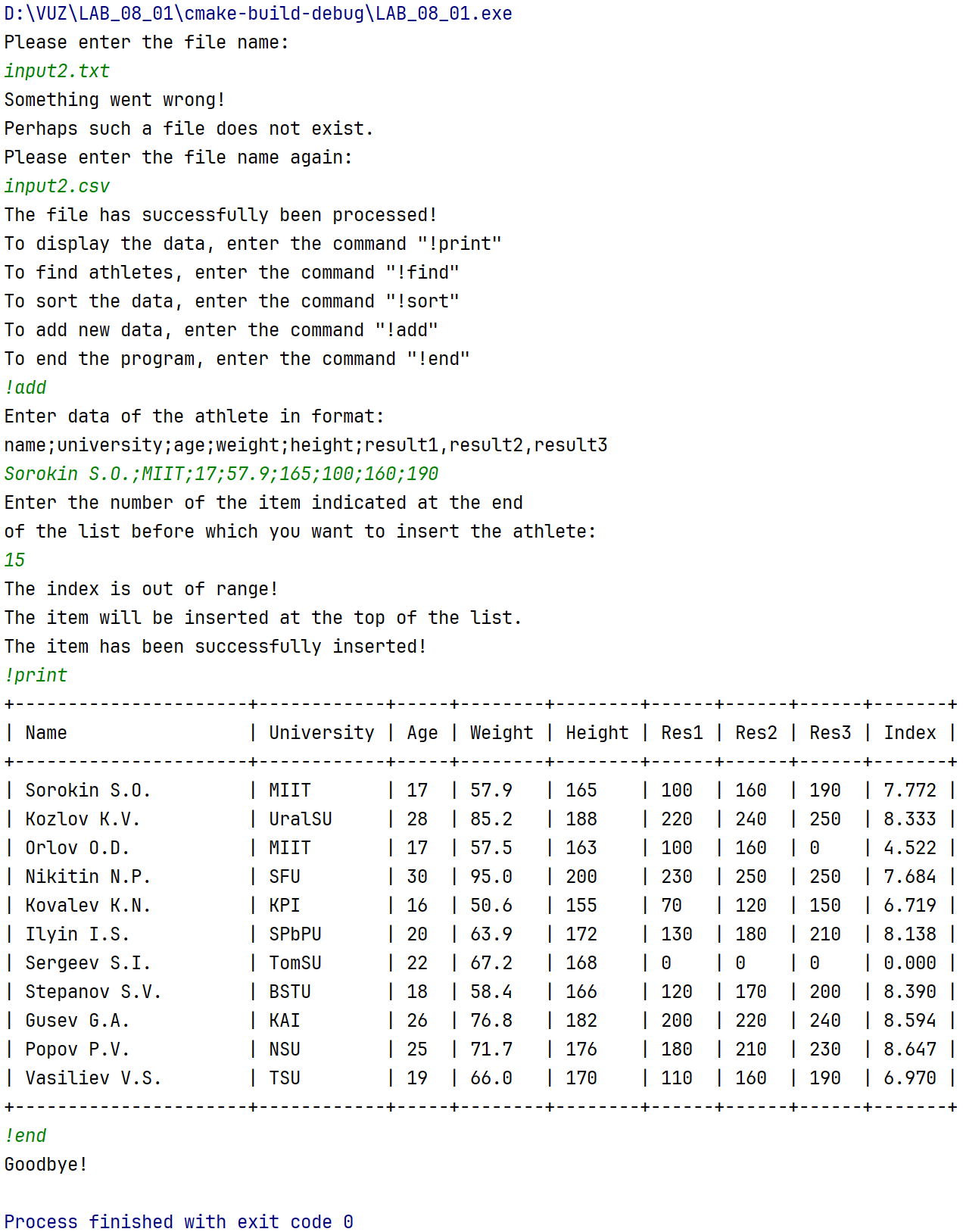


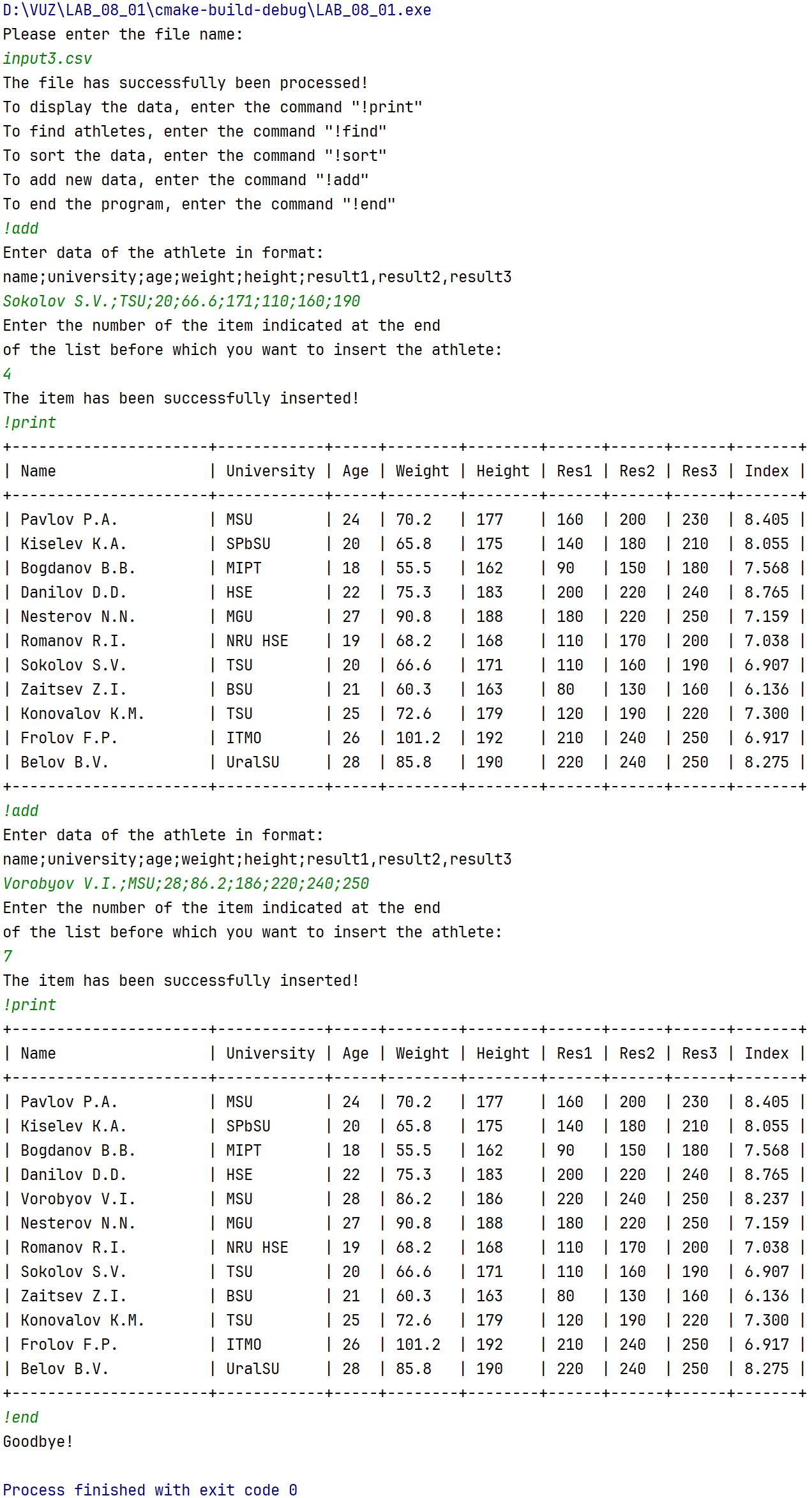




# Примеры выполнения программы







# Выводы

В результате выполнения работы изучены особенности работы с линейными односвязными списками в языке Си. А также получены практические навыки в работе с указателями на структуры.