Министерство науки и образования Украины  
Днепропетровский национальный университет им. О. Гончара

**Курсовая работа**

по программированию

Исполнитель:  
студент гр. ПК-11-1з  
Шевцов Леонид Анатольевич

2012 г.

# Оглавление

Оглавление 2

Введение 3

Постановка задачи 3

Используемые программные средства 3

Формат входных данных 4

Анализ требований и разработка алгоритма 5

Пример работы программы 6

Псевдокод 7

Выводы 8

Использованная литература 9

Примечание 1. Исходный текст программы 10

# Введение

Целью данной работы является закрепление и проверка знаний, полученных на курсе «Основы программирования на С/С++».

В частности, работа уделяет внимание:

* Чтению и записи текстовых файлов.
* Структурным типам С.
* Строковому типу и строковым операторам.
* Динамическому выделению и освобождению памяти.
* Написанию алгоритмов с использованием условий и циклов.
* Модульному программированию.

# Постановка задачи

Написать программу, которая соответствует следующим требованиям:

1. Наличие описанного извне шаблона структуры name с двумя элементами: строкой для имени и строкой для фамилии.
2. Наличие описанного извне шаблона структуры «пассажир» с элементами: структурой name, ном\_рейса (строка), кол\_вещей(целый тип), общ\_вес(целый тип).
3. Функция main() должна объявлять массив структур «пассажир» в количестве S\_SIZE (первая строка входного файла) и выполнять инициализацию структуры «пассажир» из входного файла. Определить функции для выполнения заданий, описанных ниже, результаты поместить в выходной файл. Имена входного и выходного файла считывать с командной строки.
4. Определить:
   1. Рейс с наибольшим весом багажа.
   2. Пассажира с наибольшим количеством вещей.
   3. Список пассажиров и информацию про багаж, для заданного рейса.

# Используемые программные средства

Программа писалась на языке C с использованием редактора Sublime Text 2, компилятора GNU C версии 4.2.1, отладчика GDB, утилит для сборки CMake и GNU Make на операционной системе OS X 10.8.

В дальнейшем работа программы была проверена в среде разработки Visual C++ 2010 Express на операционной системе Windows 7. Компилятором Visual C++ был создан исполнимый файл, предоставляемый вместе с данной работой.

Для контроля версий и обмена исходным кодом использовалась система контроля версий Git.

# Формат входных данных

Условие задачи упоминает, что исходные данные о пассажирах должны храниться в текстовом файле, и в первой строчке этого файла должно быть записано число пассажиров.

Остаток файла занимают записи о пассажирах.

Для простоты, положим, что в каждой строчке файла записано по одному полю структуры «Пассажир», т.е. формат файла таков:

* Число пассажиров (целое число)
* Имя первого пассажира (строка)
* Фамилия –«- (строка)
* Номер рейса –«- (строка)
* Число предметов багажа –«- (целое число)
* Общий вес багажа –«- (целое число)
* Имя второго пассажира (строка)
* … и т.д.

Все строковые поля **должны** быть не пустыми.

Все числовые поля **должны** быть больше либо равны нулю.

Пробелы вокруг полей **не допускаются**.

Число строчек в файле **должно** быть не менее 1 + (число пассажиров)\*5.

Остаток строчек в файле могут игнорироваться программой.

# Анализ требований и разработка алгоритма

При вводе данных из файла необходимо убедиться, что файл имеет правильный формат.

В первую очередь, число пассажиров должно быть не меньше нуля для соответствия формату входного файла.

Кроме того, несмотря на то, что в общем случае входной файл с нулевым числом пассажиров имеет смысл, ни одну из поставленных задач нельзя решить без наличия хотя бы одного пассажира; таким образом, проверим, что число пассажиров больше нуля.

Кроме того, нужно проверять, что строковые поля в структуре «Пассажир» не пустые, а числовые – не меньше нуля.

После введения данных и проверки их правильности, можно приступить к решению задачи.

Чтобы выполнить пункты «a» и «с» условия, будет удобно разбить список пассажиров по рейсам.

Так или иначе, для выполнения первого пункта необходимо подсчитывать вес каждого рейса, т.е. заводить структуру данных для хранения соответствия рейса и веса; однако, поскольку в дальнейшем нам придется выбрать всех пассажиров рейса, почему бы не начать с построения соответствия «рейс – пассажиры»?

После того, как такое соответствие построено, остается:

* Для пункта «а» - обойти все рейсы, подсчитать вес и найти максимальный.
* Для пункта «с» - выбрать из карты соответствия нужный нам рейс.

Построение дополнительной структуры данных для соответствий требует её правильного освобождения в конце работы.

Решение пункта «b» условия – нахождение пассажира с наибольшим количеством вещей – заключается в простом переборе пассажиров.

# Пример работы программы

## Командная строка

passengers passengers.txt output.txt

## Входной файл (passengers.txt)

3

Bill

Gates

101-B

3

90

Steve

Jobs

101-B

2

30

Steve

Ballmer

1034-C

5

100

## Выходной файл (output.txt)

Passenger with max items: Steve Ballmer (5 items)

Flight with maximum weight: 101-B

Flight information for flight no. 101-B

Total weight: 120

List of passengers

Name | B. weight

Bill Gates | 90

Steve Jobs | 30

# Псевдокод

## mapFlights (построитьОтображениеРейсов)

* Дано: Пассажиры
* Рейсы = ∅
* Для каждого пассажира из Пассажиров:
  + Искомый рейс = ?
  + Для каждого рейса из Рейсов:
    - Если номер рейса = номер рейса пассажира, то искомый рейс := рейс пассажира
  + Если искомый рейс = ?
    - Добавить новый рейс в Рейсы;
    - номер рейса := номер рейса пассажира
    - пассажиры рейса := ∅
    - искомый рейс := новый рейс
  + Добавить пассажира в пассажиров искомого рейса.
* Вернуть: Рейсы

## Другие процедуры

Все остальные места программы не представляют алгоритмической сложности и являются обыкновенными обходами массивов по циклу, иногда – с проверкой некоторого условия.

# Выводы

Перед этой работой лично у меня были сомнения насчет того, а стоит ли мириться с врожденной сложностью языка С++, либо языка С достаточно для большинства применений.

Во-первых, оказалось, что в С отсутствует возможность передачи аргумента по ссылке. Это эмулируется передачей указателя вместо аргумента, ценой усложнения кода программы.

Во-вторых, были совершены попытки применить функциональный стиль программирования к языку C, а именно, карринг, и сделаны выводы, что это невозможно либо непрактично.

В-третьих, в процессе работы было обнаружено, что среда разработки Visual Studio никаких версий не поддерживает стандарт С99 (и не планирует), а ограничивается устаревшим С89/С90.

Одна из неприятно неподдерживаемых возможностей – это объявление переменных в любом месте функции, а не только в её начале.

Из этого сделан вывод, что при написании программ на С нужно пользоваться другими компиляторами, например, GCC или, на Windows, MinGW.

Другой сделанный вывод – не стоит вообще использовать С для программ, которые будут использоваться на Windows.

Перефразируя Десятое правило Гринспена, *«каждая достаточно сложная программа на С содержит глючную, неполную и медленную реализацию языка С++»*.

Помимо этого, были попробовано использовать GDB для отладки, и сделан вывод, что GDB не так и сложен в понимании, как это рисует воображение.

# Использованная литература

* «Is there a way to do currying in C?» <http://stackoverflow.com/questions/1023261/is-there-a-way-to-do-currying-in-c>
* Laurent Dami, «More functional reusability in C/C++/Objective-C with Curried Functions» <http://asg.unige.ch/site/papers/Dami91a.pdf>
* «Visual Studio support for new C/C++ standards?» <http://stackoverflow.com/questions/146381/visual-studio-support-for-new-c-c-standards>
* «Passing by reference in C» <http://stackoverflow.com/questions/2229498/passing-by-reference-in-c>
* <http://ru.wikipedia.org/wiki/Десятое_правило_Гринспена>
* <http://en.wikipedia.org/wiki/C99>

# Примечание 1. Исходный текст программы

## main.cpp

#include "passenger\_structures.h"  
#include "passenger\_input.h"  
#include "passenger\_logic.h"  
  
#include <stdlib.h>  
#include <stdio.h>  
  
void printBanner() {  
 printf("Not enough arguments\nUsage: passengers [input\_filename] [output\_filename]\n");  
}  
  
void readData(const char\* filename, Passenger\*\* passengers, int\* passengers\_size) {  
 FILE\* input\_file = fopen(filename, "r");  
 \*passengers = readPassengerFile(input\_file, passengers\_size);  
 fclose(input\_file);  
}  
  
void printResults(const char\* filename, Passenger\* max\_passenger, Flight\* max\_flight) {  
 FILE\* output\_file = fopen(filename, "w");  
 fprintf(output\_file, "Passenger with max items: %s %s (%i items)\n", max\_passenger->name.first\_name, max\_passenger->name.last\_name, max\_passenger->items\_count);  
 fprintf(output\_file, "Flight with maximum weight: %s\n", max\_flight->flight\_no);  
 printFlight(output\_file, max\_flight);  
 fclose(output\_file);  
}  
  
int main(int argc, const char\*\* argv) {  
 // Initialization  
 if (argc < 3) {  
 printBanner();  
 return -1;  
 }  
  
 Passenger \*passengers = NULL, \*max\_passenger = NULL;  
 Flight \*flights = NULL, \*max\_flight = NULL;  
 int passengers\_size = 0, flights\_size = 0;  
  
 readData(argv[1], &passengers, &passengers\_size);  
  
 // Error handling  
 if (passengers == NULL) {  
 printf("Incorrect passenger file format.\n");  
 return -1;  
 }  
 if (passengers\_size == 0) {  
 printf("Passenger file must contain at least one record.\n");

freePassengers(passengers);  
 return -1;  
 }  
  
 // Core logic  
 max\_passenger = passengerWithMaxItems(passengers, passengers\_size);  
 flights = mapFlights(passengers, passengers\_size, &flights\_size);  
 max\_flight = flightWithMaxWeight(flights, flights\_size);  
  
 // Output  
 printResults(argv[2], max\_passenger, max\_flight);  
  
 // Finalization  
 freeFlights(flights, flights\_size);  
 freePassengers(passengers, passengers\_size);  
 return 0;  
}

## passenger\_input.h

#ifndef \_\_PASSENGER\_INPUT\_H\_\_  
#define \_\_PASSENGER\_INPUT\_H\_\_  
  
#include <stdio.h>  
#include "passenger\_structures.h"  
  
Passenger\* readPassengerFile(FILE\* input\_file, int\* size);  
int readPassenger(FILE\* input\_file, Passenger\* passenger);  
  
#endif

## passenger\_input.cpp

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <ctype.h>  
#include "passenger\_structures.h"  
#include "passenger\_input.h"  
  
// Helper function to read integer with proper error handling  
int readInt(FILE\* file, int \*i) {  
 char buffer[100];  
 char \*ptr;  
 if (fgets(buffer, sizeof(buffer), file) != NULL) {  
 (\*i) = strtol(buffer, &ptr, 10);  
 return buffer[0] != '\n' && (\*ptr == '\n' || \*ptr == '\0');  
 } else {  
 return 0;  
 }  
}  
  
int readLine(FILE\* file, char\*\* line) {  
 char buffer[100];  
 if (fgets(buffer, sizeof(buffer), file) != NULL) {  
 // Trim trailing newline & spaces  
 int buf\_len = strlen(buffer)-1;  
 while(buf\_len>0 && isspace(buffer[buf\_len])) buf\_len--;  
 buffer[buf\_len+1] = 0;  
 (\*line) = new char[buf\_len+1];  
 strcpy((\*line), buffer);  
 return 1;  
 } else {  
 return 0;  
 }  
}  
  
int readName(FILE\* file, Name\* name) {  
 return readLine(file, &name->first\_name) && readLine(file, &name->last\_name);  
}  
  
// TODO while it would be nice to provide custom error messages,  
// this functionality is out of scope of this coursework  
int validatePassenger(Passenger\* passenger) {  
 return  
 (strlen(passenger->name.first\_name) > 0) &&  
 (strlen(passenger->name.last\_name) > 0) &&  
 (strlen(passenger->flight\_no) > 0) &&  
 (passenger->total\_weight >= 0) &&  
 (passenger->items\_count >= 0);  
}  
  
Passenger\* readPassengerFile(FILE\* input\_file, int\* size) {  
 if (!readInt(input\_file, size)) {  
 return NULL; // bad file format  
 }  
 if ((\*size)<0) {  
 return NULL;  
 }  
 Passenger\* passengers = new Passenger[\*size];  
 int i;  
 for (i=0; i<(\*size); ++i) {  
 if (!readPassenger(input\_file, passengers+i)) {  
 // FIXME there is a known memory leak of already allocated strings in the last Passenger instance  
 // Fix omitted here for clarity  
 freePassengers(passengers, i);  
 return NULL;  
 }  
 if (!validatePassenger(passengers+i)) {  
 // The current passenger must also be disposed of  
 freePassengers(passengers, i+1);  
 return NULL;  
 }  
 }  
 return passengers;  
}  
  
int readPassenger(FILE\* input\_file, Passenger\* passenger) {  
 return readName(input\_file, &passenger->name) &&  
 readLine(input\_file, &passenger->flight\_no) &&  
 readInt(input\_file, &passenger->items\_count) &&  
 readInt(input\_file, &passenger->total\_weight);  
}

## passenger\_logic.h

#ifndef \_\_PASSENGER\_LOGIC\_H\_\_  
#define \_\_PASSENGER\_LOGIC\_H\_\_  
  
#include <stdio.h>  
#include "passenger\_structures.h"  
  
Flight\* mapFlights(Passenger\* passengers, const int size, int \*flights\_size);  
Flight\* flightWithMaxWeight(Flight\* flights, const int size);  
Passenger\* passengerWithMaxItems(Passenger\* passengers, const int size);  
void printFlight(FILE\* output\_file, Flight\* flight);  
  
#endif

## passenger\_logic.cpp

#include <stdlib.h>  
#include <string.h>  
#include "passenger\_logic.h"  
  
Flight\* mapFlights(Passenger\* passengers, const int size, int \*flights\_size) {  
 // Yep, not very effective. But this is the max possible number of flights  
 Flight\* flights = new Flight[size];  
 \*flights\_size = 0;  
 int i, j;  
  
 for (i=0; i < size; ++i) {  
 for (j=0; j < \*flights\_size; ++j) {  
 if (strcmp(passengers[i].flight\_no, flights[j].flight\_no)==0) {  
 break;  
 }  
 }  
 if (j == \*flights\_size) { // didn't find a flight in the map; initialize new  
 flights[j].flight\_no = passengers[i].flight\_no;  
 flights[j].passenger\_count= 0;  
 flights[j].passengers = new Passenger\*[size]; // also not very effective  
 (\*flights\_size)++;  
 }  
 flights[j].passengers[flights[j].passenger\_count++] = passengers+i;  
 }  
  
 return flights;  
}  
  
int weightOfFlight(Flight\* flight) {  
 int i, weight = 0;  
  
 for (i=0; i < flight->passenger\_count; ++i) {  
 weight += flight->passengers[i]->total\_weight;  
 }  
  
 return weight;  
}  
  
Flight\* flightWithMaxWeight(Flight\* flights, const int size) {  
 int i, max\_weight = 0;  
 Flight\* max\_flight = NULL;  
  
 for (i=0; i < size; ++i) {  
 int flight\_weight = weightOfFlight(flights+i);  
 if (flight\_weight > max\_weight) {  
 max\_weight = flight\_weight;  
 max\_flight = flights + i;  
 }  
 }  
  
 return max\_flight;  
}  
  
Passenger\* passengerWithMaxItems(Passenger\* passengers, const int size) {  
 int i, max\_count = 0;  
 Passenger \*max\_passenger = NULL;  
  
 for (i=0; i < size; ++i) {  
 if (passengers[i].items\_count > max\_count) {  
 max\_count = passengers[i].items\_count;  
 max\_passenger = passengers + i;  
 }  
 }  
  
 return max\_passenger;  
}  
  
char\* fullName(Name\* name) {  
 char\* full\_name = new char[strlen(name->first\_name) + strlen(name->last\_name) + 2];  
 strcpy(full\_name, name->first\_name);  
 strcat(full\_name, " ");  
 strcat(full\_name, name->last\_name);  
 return full\_name;  
}  
  
void printFlight(FILE\* output\_file, Flight\* flight) {  
 int i;  
  
 fprintf(output\_file, "Flight information for flight no. %s\n", flight->flight\_no);  
 fprintf(output\_file, "Total weight: %i\n", weightOfFlight(flight));  
 fprintf(output\_file, "List of passengers\n%20s | %10s\n", "Name", "B. weight");  
  
 for (i=0; i < flight->passenger\_count; ++i) {  
 char\* full\_name = fullName(&flight->passengers[i]->name);  
 fprintf(output\_file, "%20s | %10i\n", full\_name, flight->passengers[i]->total\_weight);  
 free(full\_name);  
 }  
}

## passenger\_structures.h

#ifndef \_\_PASSENGER\_STRUCTURES\_H\_\_  
#define \_\_PASSENGER\_STRUCTURES\_H\_\_  
  
typedef struct {  
 char\* first\_name;  
 char\* last\_name;  
} Name;  
  
typedef struct {  
 Name name;  
 char\* flight\_no;  
 int items\_count;  
 int total\_weight;  
} Passenger;  
  
typedef struct {  
 const char\* flight\_no;  
 int passenger\_count;  
 Passenger\*\* passengers;  
} Flight;  
  
void freeFlights(Flight\* flights, const int size);  
void freePassengers(Passenger\* passengers, const int size);  
  
#endif

## passenger\_structures.cpp

#include <stdlib.h>  
#include "passenger\_structures.h"  
  
void freeFlights(Flight\* flights, const int size) {  
 int i;  
 for (i=0; i < size; ++i) {  
 free(flights[i].passengers);  
 }  
 free(flights);  
}  
  
void freePassengers(Passenger\* passengers, const int size) {  
 int i;  
 for (i=0; i < size; ++i) {  
 free(passengers[i].name.first\_name);  
 free(passengers[i].name.last\_name);  
 free(passengers[i].flight\_no);  
 }  
 free(passengers);  
}