Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет непрерывного и дистанционного обучения

Кафедра экономической информатики

Контрольная работа №1
По курсу «Основы алгоритмизации и программирования» часть 1

Выполнил — Студент
Ф.И.О. Юруш Е.В.
№ зач. кн. (нет)

Проверил — Т. М. Унучек

Минск-2017

Содержание

Введение

Выполнение данной контрольной работы заключается в разработке консольного приложения для автоматизации учёта студентов в общежитии с использованием языка программирование C, Её цель - овладеть начальными практическими навыками программирования на языке C и работы в среде программирования Visual C++, разбор структуры программы и построение её алгоритма, формирование компьютерной грамотности.

1. Теоретические вопросы

Алфавит языка. Идентификаторы.

Алфавит языка - это совокупность символов, используемых в языке. Символы, которые могут быть использованы в языке Си включают в себя:

1. буквы латинского алфавита
2. цифры (0-9)
3. специальные знаки (“ ‘ {} [] () | = + - * / % \ ; . , : ? < > _ ! & # ~ ^)
4. пробельные символы (пробел, табуляция, переход на новую строку)

Компилятор языка Си различает большие и малые латинские буквы. Буквы русского алфавита доступны к использованию в комментариях

Идентификатор в языке Си – это последовательность букв, цифр и символов подчёркивания, начинающаяся с буквы или символа подчёркивания.

Особенности идентификаторов:

1. компилятор учитывает первые 31 символ от начала идентификатора (в некоторых компиляторах только 8)
2. идентификаторы должны начинаться с буквы или символа подчеркивания
3. собственные идентификаторы не должны совпадать с другими идентификаторами и ключевыми словами.

Идентификаторы делятся на:

1. ключевые слова - идентификаторы, которые используются только по своему назначению (напр. int).
2. зарезервированные слова - это идентификаторы, которые имеют определенный смысл в языке, назначение которых можно изменить (напр printf).
3. собственные идентификаторы – это идентификаторы, которые программист объявляет в программе для своих целей.

Другие директивы препроцессора: #if, #ifdef, #ifndef, #else, #endif.

Директивы #if, #elif, #else, #endif относятся к директивам условной компиляции.

Синтаксис их использования следующий:

#if константное_выражение 1

фрагмент_текста 1

#elif константное_выражение 2

фрагмент_текста 2

…

#else

фрагмент текста n

#endif

С данной директивой прекомпилятор работает по следующему принципу: поочерёдно сверху вниз анализируются константные_выражения до выражения результат которого - ненулевой результат, в случае нахождения такого к компиляции допускается соответствующий данному константному выражению фрагмент_текста.

Также к условным директивам компиляции относят директивы: #ifdef, #ifndef. #ifdef означает «определено», #ifndef соответственно «не определено». Синтаксис их использования следующий:

#ifdef макро_имя

фрагмент_текста 1

#else

фрагмент текста 2

#endif

Если директивой #define было ранее определено макро_имя, тогда будет оставлен фрагмент_текста 1, иначе оставлен будет фрагмент_текста 2. Если #ifdef заменить на #ifndef директива будет работать обратным образом.

2. Практическая часть

Задание на разработку программы в соответствии с индивидуальным заданием: в соответствии с вариантом индивидуального задания, необходимо разработать консольное приложение на языке С. Приложение должно предоставлять возможности: просмотра информации из текстового файла; добавления новых записей в файл; удаления записей из файла; редактирования записей в файле. В работе предусмотреть использование пользовательских функций, массивов, структур. Вариант индивидуального задания 1, предметная область «Учет студентов в общежитии».

Листинг кода программы:

Создаём заголовочный файл header.h. В нём:

```c
//объявляем символьную константу в которой храним количество символов
//в поле фио студента структуры Student
#define MAXFNAME 25
```

```c
//объявляем структуру
typedef struct tagStudent
{
    int    studNmb;          //номер в списке
    char   fname[MAXFNAME]; //фио студента
    int    roomNmb;          //номер комнаты
    int    floor;            //этаж

    //в этих переменных храним ссылку на следующую, предыдущую связанную структуру
    struct tagStudent *pnext, *pprev;
} Student;
```

```c
//объявляем прототипы функций используемых в программе
void print(Student *p);
void getUser(Student *p);
Student* addEnd(Student *p, Student *end, int stnumb);
void printList(Student *p);
void loadList(Student *p, char *file, Student **pend, Student **pbegin);
void saveList(Student *p, char *file);
int editStd(Student *p, Student **pend, Student **pbegin);
int deleteStd(Student *p, Student **pend, Student **pbegin);
```

Создаём главный файл main.c. В нём:

```c
//подключаем необходимые заголовочные файлы стандартной библиотеки
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
//подключаем созданный заголовочный файл
#include "header.h"
```

```c
//пишем главную функцию программы
void main()
{
    //объявляем переменные
    //переменная для выбранного пользователем номера команды
    int n=0;
    //перем. для временного хранения стр-ры, для передачи её из одной функции в другую
    Student std;
    //указатели на первый и последний элемент связанных структур
    Student *begin=NULL, *end=NULL;
```

Организуем пользовательское меню.

```c
    //пользовательское меню
    //1.добавить запись в список студентов 2.отредактировать запись в списке
    //3.сохр. список в файл 4.загруз. список из файла 5.вывести список на экран
    //6.выход
L:  printf("\n1.add    3.save    5.print  \n2.edit    4.load    6.exit    \n");
    printf("Input the command number (from 1 to 6): ");
    //сохраняем введённое пользователем значение в переменной n
    scanf("%d", &n);

    //в зав-ти от введённого пользователем значения (1-5) запуск. соотв. функции,
    //при вводе '6' (6й команды) выходим без ошибки
    switch(n)
    {
    case 1: //принимаем значения от пользователя во временную структуру std
            getUser(&std);
            //связываем заполненную польз-ем стр-ру std с имеющимися структурами
            end=addEnd(&std,end,0);
            //при первом заполнении стр-ры указатели на 1 и последн. структуру равны
            if (begin==NULL) begin=end;
            break;
    case 2: //редактируем выбранную пользователем строку
            editStd(begin, &end, &begin);
            break;
    case 3: //сохраняем список в файл
            saveList(begin,"list.dat"); break;
    case 4: //загружаем список из файла
            loadList(begin,"list.dat",&end, &begin); break;
    case 5: //печатаем список на экран
            printList (begin); break;
    case 6: //завершаем программу без ошибки
            exit(0); break;
    }
    //при вводе пользователем значения не равного 1,2,3,4,5,6
    //просим пользователя ввести команду ещё раз
    goto L;
}
```

Опишем функции которые запускаются при вводе команды добавления записи в список ‘1’.

```
48  //функция для приёма значений от пользователя
49  //принимает параметром указатель на структуру
50  void getUser(Student *p)
51  {
52      Student tmp;
53      //поочерёдно запрашиваем и принимаем информацию от пользователя
54      printf("\nInput name :");
55      fflush(stdin);
56      fgets(tmp.fname,MAXFNAME,stdin);
57      //удаляем символ завершения строки кот. добавился функцией fgets
58      tmp.fname[strlen(tmp.fname)-1]='\0';
59      printf("Input floor: ");
60      scanf("%d", &tmp.floor);
61      fflush(stdin);
62      printf("Input room number: ");
63      scanf("%d", &tmp.roomNmb);
64
65      //указатели на пред. след. структуру укажем позже
66      tmp.pnext = tmp.pprev = NULL;
67      //копируем структуру tmp в структуру по указателю p (вход. пар-р)
68      *p = tmp;
69  }
```

```
79  //функция (ф) для связывания заполненной польз-ем структуры с имеющимся списком
80  //структур. ф принимает: заполненную структуру, указатель на последнюю
81  //структуру в списке, номер в списке который необходимо заполнить для
82  //добавляемого студента (т.к. данная ф-я еще используется при загрузке
83  //списка из файла). ф возвращает: указатель на структуру который записывается в
84  //переменную end - указатель на последнюю структуру
85  Student* addEnd(Student *p, Student *end, int stnumb)
86  {
87      //выделяем память для структуры, помещаем указатель на неё в pAdd
88      Student *pAdd=(Student*)malloc(sizeof(Student));
89      //заполняем память по адресу pAdd переданной, через параметр указатель p, структурой
90      *pAdd=*p;
91      if (end==NULL) //если добавляем первый элемент
92      {
93          //в указатель end помещаем указатель на заполненную структуру
94          end=pAdd;
95          //если переданный параметр stnumb равен нулю, тогда номеру в списке присваиваем 1
96          //иначе значение равное переданному параметру stnumb
97          if (stnumb==0) pAdd->studNmb=1;
98          else pAdd->studNmb=stnumb;
99      }
100     else //если добавляем не первый элемент
101     {
102         end->pnext = pAdd;//связываем последний элемент с добавляемым
103         pAdd->pprev=end;  //связываем добавляемый элемент с последним
104         //если переданный параметр stnumb равен нулю, тогда номер в списке присваиваем
105         //номер последней строки в списке плюс один
106         if (stnumb==0) pAdd->studNmb=end->studNmb+1;
107         else  pAdd->studNmb=stnumb;
108         end= pAdd;
109     }
110     //возвращаем указатель на последний элемент
111     return end;
112 }
```

Опишем функции, которые запускаются при вводе команды редактирования записи в списке ‘2’.

```
113  //функция для редактирования выбранных пользователем строк
114  //вход. пар-ры: указатель на первую структуру,
115  //указатель на указатель последней в списке структуры,
116  //указатель на указатель первой в списке структуры,
117  int editStd(Student *p, Student **pend, Student **pbegin)
118  {   //объявляем переменные
119      int n, k;
120      Student *s, std;
121      //выводим список на экран и предлагаем выбрать номер строки
122      printList (p);
123      printf("Input number of student: ");
124      scanf("%d", &n);
125      //находим структуру номер в списке кот. равен выбранному
126      while(p!=NULL) {
127      if (p->studNmb==n){ s=p; break;}
128      p=p->pnext; }
129      //предлагаем выбрать номер команды "что сделать со строкой"
130  L:  printf("\nWhat do you want to do with it? \n1.edit \n2.delete\
131   \n3.exit to the main menu \nInput the command number : ");
132      scanf("%d", &k);
133      switch(k)
134      {
135      case 1: //при редактировании строки запрашиваем перевввести данные
136                  getUser(&std);//и сохраняем их в структуру std
137                  //в выбранную пользователем строку вносим изменения
138                  strncpy(s->fname,std.fname,sizeof(std.fname));
139                  s->roomNmb=std.roomNmb; s->floor=std.floor;
140                  goto M; //выход из цикла
141      case 2: //удаляем строку функцией deleteStd, передаём в неё
142                  //указатель на удаляемую стр-ру, на первый и последний эл-ты списка
143                  deleteStd(s, pend, pbegin); goto M; //и выходим из цикла
144      case 3: goto M; //выход из цикла в главное меню
145      }
146      goto L;  //переспрашиваем номер команды если введённой значение не 1,2,3
147  M:  return 0;//выходим без кода об ошибке
148  }
```

Опишем функцию по указателю deleteStd.

```
149  //удаление строки из списка, параметры идентичные функции editStd
150  int deleteStd(Student *p, Student **pend, Student **pbegin)
151  {
152      Student *pr=NULL, *pn=NULL;//указатели для пред. след. эл. списка
153      if (p->pnext==NULL) //если удаляемая строка последняя
154      {
155          pr=p->pprev; //сохраняем в pr указатель на предыд. эл. списка
156          if(pr!=NULL) //если предыд. эл. есть
157          {
158              pr->pnext=NULL; //обнуляем указатель в пред. эл-те
159              *pend=pr;       //предыдущий элемент объявляем последним
160
161          } else //если предыд. элемента нет (т.е. элемент в списке один)
162          {
163              *pend=NULL; //последний и первый элементы списка обнуляем
164              *pbegin=NULL;
165          }
166      }
167      else if (p->pprev==NULL) //если удаляемая строка первая
168      {
169          pn=p->pnext; //сохраняем в pn указатель на след. эл. списка
170          *pbegin=pn;  //указатель на первый элемент меняем на pn
171          pn->pprev=NULL; //в след эл-те обнуляем указатель на пред. эл-т
172      }
173      else //если удаляемая стр. ни первая ни последняя
174      {
175          pn=p->pnext; //сохраняем в pn указатель на след. эл. списка
176          pr=p->pprev; //сохраняем в pr указатель на предыд. эл. списка
177     //связываем указ-ли предыдущ. и след. эл-та так как тек. эл-т удаляем
178          pn->pprev=p->pprev;
179          pr->pnext=p->pnext;
180      }
181      free(p); //освобождаем память по указателю тек. элемента
182      p=NULL;  //обнуляем указатель
183      return 0; //выходим без кода ошибки
184  }
```

Опишем функцию, которая запускается при вводе команды сохранения списка в файл ‘3’.

```
187  void saveList(Student *p, char *file)
188  {
189      //объявляем указатель pf типа FILE, открываем/создаём файл
190      //функцией fopen (режим символьн. записи), и помещаем указатель на файл в pf
191      FILE *pf=fopen(file,"w");
192      //если файл успешно открыт/создан
193      if (pf!=NULL)
194      {
195          while(p!=NULL)//проходимся по всем элементам списка
196          {
197          //и записываем элементы структур в файл
198          fprintf(pf,"%d %s %d %d\n", p->studNmb, p->fname, p->roomNmb, p->floor);
199          p=p->pnext;
200          }
201          fclose(pf);//закрываем файл
202      }
203  }
```

Опишем функцию, которая запускается при вводе команды загрузки списка из файла ‘4’.

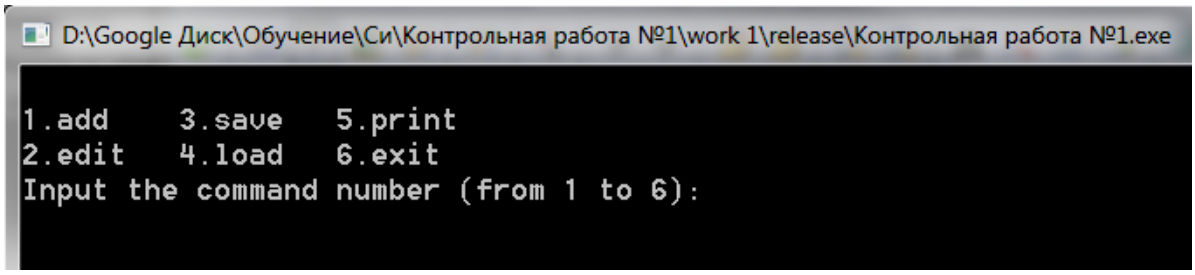
```
205  //функция загрузки списка из файла, параметры:
206  //указатель на структуру типа Student, название файла, указатель на
207  //указ-ль на первый эл-т списка, указ-ль на указ-ль на последний эл-т списка
208  void loadList(Student *p, char *file, Student **pend, Student **pbegin)
209  {
210      Student tmp, *ptmp;//
211      //объявляем указатель pf типа FILE, открываем/создаём файл
212      //функцией fopen (режим символьн. чтения), и помещаем указатель на файл в pf
213      FILE *pf=fopen(file,"r");
214      //если файл успешно открыт/создан
215      if (pf!=NULL)
216      {
217          //очищаем текущий список структур
218          while(p!=NULL)
219          {
220              ptmp=p->pnext;
221              deleteStd(p, *pend, *pbegin);
222              p=ptmp;
223          }
224          *pend=NULL;
225          *pbegin=NULL;
226          //пока не достигнут конец файла читаем данные во временную структуру tmp
227          while(!feof(pf))
228          {
229              fscanf(pf,"%d %s %d %d\n", &tmp.studNmb, &tmp.fname, &tmp.roomNmb, &tmp.floor);
230              tmp.pnext = tmp.pprev = NULL;
231              //запускаем функцию addEnd, передаём в неё указатель на заполненную временную структуру,
232              //указатель на указатель последнего элемента списка, и номер записываемого элемента в списке
233              //функция добавит полученную из файла структуру в список
234              *pend=addEnd(&tmp,*pend,tmp.studNmb);
235              //если элемент первый то последний элемент равен первому элементу
236              if (*pbegin==NULL) *pbegin=*pend;
237          }
238          fclose(pf);//закрываем файл
239      }
240  }
```

Опишем функцию, которая запускается при вводе команды печати списка на экран ‘5’.

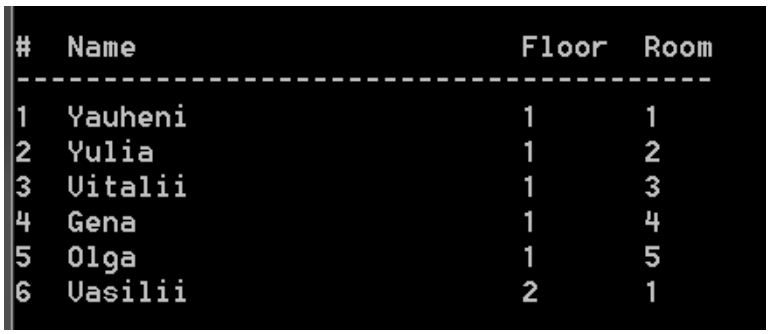
```
241  //функция вывода списка на экран принимает указатель на структуру
242  void printList(Student *p)
243  {
244      //выводим шапку таблицы
245      printf("\n%s %s %s %s \n","# ","Name                          ","Floor ","Room ");
246      printf("%s \n","----------------------------------------");
247
248      //циклом проходимся по элементам списка и выводим их на экран
249      //ниже описанной функцией print
250      while(p!=NULL)
251      {
252          print(p);
253          p=p->pnext;
254      }
255  }
256  //функция печати строки списка н экран
257  void print(Student *p)
258  {
259      printf("%-2d %-25s %-6d %-3d \n", p->studNmb, p->fname, p->floor, p->roomNmb);
260  }
```

Интерфейс работы программы:

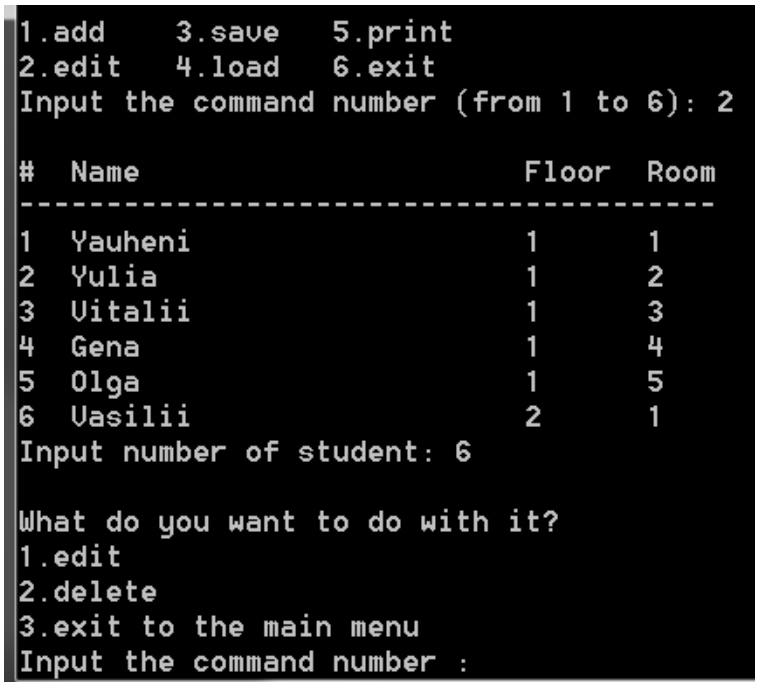
Главное меню:



Печать списка студентов:



Редактирование записи о студенте:



3. Блок-схема работы программы.

Заключение

В соответствии с индивидуальным заданием №1 разработана консольная программа учёта студентов в общежитии. Предоставляется возможность вести список студентов, и их расположения в общежитии: добавлять, редактировать удалять строки, сохранять информацию в файл и в последующем загружать её из файла. Программа прокомментирована, разобрана её структура. Отработаны навыки работы с массивами, структурами, строками, функциями, а также отработана запись информации в файл и её чтение из файла. Предоставлены ответы на теоретические вопросы.

Литература

1. Подбельский В.В., Фомин С.С. Программирование на языке Си: Учебное пособие. 2-е доп. изд. - М.: Финансы и статистика – 600 с.: ил.
2. Березин Б.И., Березин С.Б. Начальный курс С и С++. –М.: Диалог - МИФИ, 1999 - 288 с.
3. Основы алгоритмизации и программирования. Язык Си: учеб. пособие / М. П. Батура, В. Л. Бусько, А. Г. Корбит, Т. М. Кривоносова. - Минск: БГУИР, 2007. - 240 с.: ил.