Панчоян Сергей СКБ223_2. Отчёт по ЛР#5

Файлы:

1. main.cpp:
   В нём проверяются введённые ключи и вызывается основная функция.
2. funcs.cpp:
   Файл, в котором находятся все необходимые функции, исполняемые программой.
3. funcs.h:
   Заголовочный файл. Используется в main.cpp. содержит некоторые функции из funcs.cpp.
4. makefile:
   Файл make. Поддерживает команды make, clean и distclean.
5. input.txt:
   Опциональный файл. Можно использовать как файл ввода при использовании соответствующего ключа.

Функционал:

1. При вызове .exe файла можно использовать 2 доступных ключа:
   1.1. “--fromfile”
   Использует данные из файла. Если после ключа ввести полный путь к нужному файлу, то будет использован он, иначе будет использован файл по умолчанию input.txt.
   Стоит обратить внимание на то, что в пути файла не должно быть пробелов, иначе программа выдаст ошибку.
   Если не указать этот ключ, данные будет необходимо ввести непосредственно в консоль. Также при вводе необходимо указать от 2 до 10000 отрезков, иначе программа выведет ошибку.
   1.2. “--tofile”:
   Использует файл в качестве точки вывода данных. Если после ключа ввести полный путь к нужному файлу, то будет использован он, иначе будет создан и использован файл по умолчанию output.txt.
   Стоит обратить внимание на то, что если файл output.txt уже существует в папке с файлами, то всё его содержимое будет удалено и заменено на новые данные. Также, если в указанном пути будут присутствовать пробелы программа выдаст ошибку

2. Вызов makefile допускает 3 основные команды.
   2.1. “make”:
   Эта команда транслирует файлы main.cpp и funcs.cpp в соответствующие объектные файлы и компанует из в main.exe. Если при вызове makefile не указать команду, то по умолчанию будет вызвана команда make.
   2.2. “clean” (будет работать только для ОС Windows):
   Эта команда удаляет созданные командой make объектные файлы.
   2.3. “distclean” (будет работать только для ОС Windows):
   Эта команда удаляет созданный командой make файл main.exe, а также созданный программой файл output.txt.

Функции:

1. Файл main.cpp:
   1.1. main( )
   В нём используются только функции описанные в файле funcs.cpp. Сама функция проверяет введённые ключи и передаёт их в качестве параметров в основную функцию.

```
1  int main (int argc, char **argv)
2  {
3    bool ff=false, tf=false, dff=false, dtf=false;
4    char *fdir="./input.txt", *tdir="./output.txt";
5    const char *key1 = "--tofile", *key2 = "--fromfile", *key = "--";
6
7    for (int i=1; i<argc; ++i)
8    {
9      if (comp(key,argv[i]))
10     {
11       if (comp(argv[i],key1))
12       {
13         if (tf) ERR
14         tf=true;
15         dtf=true; dff=false;
16       }
17       else if (comp(argv[i],key2))
18       {
19         if (ff) ERR
20         ff=true;
21         dff=true; dtf=false;
22       }
23       else
24       {
25         std::cerr << "\nWrong keys.\n";
26         return EXIT_FAILURE;
27       }
28     }
```

```
    else if (dtf!=dff)
    {
      if (dtf)
      {
        tdir = argv[i];
        dtf = false;
      }
      else if (dff)
      {
        fdir = argv[i];
        dff = false;
      }
    }
    else
    {
      std::cerr << "\nWrong keys.\n";
      return EXIT_FAILURE;
    }
  }

  func(tf, tdir, ff, fdir);

  return EXIT_SUCCESS;
}
```

2. Файл funcs.cpp:

2.1. Class Segm:

По своей сути объекты этого класса представляют из себя массив вещественных чисел длиной 2. Объект хранит в себе 2 числа: начало и конец отрезка. Также задаётся несколько функций и конструкторов для работы с этими объектами.

```
class Segm
{
    public:
      float beg;
      float end;
      Segm ();
     ~Segm ();
      void print(bool);
      void copy(const Segm&);
      bool get (bool);
};

Segm::Segm ()
  :beg(1), end(0) {};
Segm::~Segm ()
  {beg=1, end=0;}
void Segm::print(bool to)
  {
    if (to) fout << beg << " " << end << endl;
    else cout << beg << " " << end << endl;
```

```
21   }
22 void Segm::copy(const Segm& obj)
23   {beg=obj.beg; end=obj.end;}
24 bool Segm::get (bool from)
25   {
26     if(from) return ((fin >> Sget.beg)&&(fin >> Sget.end));
27     return ((cin >> Sget.beg)&&(cin >> Sget.end));
28   }
```

2.2. comp ( )

Сравнивает 2 символьных массива на то, является ли первый массив началом второго. По совместительству, этой функцией можно проверить одинаковы ли два символьных массива.

```
1  bool comp (const char *a, const char *b)
2  {
3    for (int i=0; a[i]!='\0'; ++i)
4    {
5      if (a[i]!=b[i])
6      {
7        return false;
8      }
9    }
10   return true;
11 }
```

2.3. not_correct ( )

Проверяет больше ли начало отрезка его же конца (т.е. проверяет правильно ли задан отрезок).

```
1  bool not_correct(const Segm& a)
2  {
3    if (a.end<a.beg) return 1;
4    return 0;
5  }
```

2.4. compare ( )

Сравнивает 2 отрезка и возвращает значение в зависимости от того как именно коммуницируют эти отрезки. (Первый содержит второй, только начало второго в первом, только начало первого во втором, второй содержит первый или они не пересекаются)

```
1  int compare(const Segm& a, const Segm& b)
2  {
3    if (not_correct(a)) return 0;
4    if ((a.beg<=b.beg)&&(a.end>=b.end)) return -1;
5    if ((a.beg<=b.beg)&&(a.end>=b.beg)) return 1;
6    if ((a.beg<=b.end)&&(a.end>=b.end)) return 2;
7    if ((b.beg<=a.beg)&&(b.end>=a.beg)) return 3;
8    return 0;
9  }
```

2.5. connect ( )

Если это возможно объединяет введённые отрезки в зависимости от того, как они коммуницируют.

```
bool connect(Segm& a, Segm& b)
{
  switch (compare (a,b))
  {
    case 0: return false;
    case 1: a.end=b.end; b.beg=a.beg; return true;
    case 2: a.beg=b.beg; return true;
    case 3: a.copy(b); return true;
    default: return true;
  }
}
```

2.6. put ( )

Вставляет отрезок в массив отрезком таким образом, чтобы их начала шли по возрастанию.

```
void put ()
{
  int flag=0; Segm buf1, buf2;
  for (int i=0; i<ns; ++i)
  {
    if (not_correct(m[i])) {flag=1; break;}
    if (flag)
    {buf2.copy(m[i]); m[i].copy(buf1); buf1.copy(buf2);}
    if ((Sget.beg<m[i].beg)&&!flag)
    {++flag; buf1.copy(m[i]); m[i].copy(Sget);}
  }
  if (flag) m[ns++].copy(buf1);
  else m[ns++].copy(Sget);
}
```

2.7. func ( )

Основная функция. Обрабатывает введённые данные с помощью циклов и вышеуказанных функций.

```
void func(bool tf, const char* tdir, bool ff, const char* fdir)
{
  unsigned int n;
  if (ff)
  {
    fin.open(fdir);
    if (!fin.is_open())
    {
      cerr << "Wrong input file directory.\n";
      exit(EXIT_FAILURE);
    }
    fin >> n;
  }
  else cin >> n;
  if(n==1)
  {
    cerr << "Nothing found.\n";
    exit(EXIT_FAILURE);
  }
  if (n>10000)
```

```
21    {
22      cerr<< "Too much segments.\n";
23      exit (EXIT_FAILURE);
24    }
25    m = new Segm[n];
26    for (int i=0; i<n; ++i)
27    {
28      int flag=0;
29      if (!Sget.get(ff))
30      {
31        cerr<< "Not enough numbers.\n";
32        exit(EXIT_FAILURE);
33      }
34      if (not_correct(Sget))
35      {
36        cerr << "Wrong segment in " << (i+1) << " line." << endl;
37        exit (EXIT_FAILURE);
38      }
39      for (int j=0; j<ns; ++j)
40      {
41        if(connect(m[j],Sget)) flag =1;
42      }
43      if (!flag) put();
44    }
45    for (int i=1; i<ns; ++i)
46    {
47      if (m[i-1].beg==m[i].beg) m[i-1].end=m[i-1].beg-1;
48    }
49
50    if(tf)
51    {
52      fout.open(tdir,std::ofstream::trunc);
53      if (!fout.is_open())
54      {
55        cerr << "Wrong output file directory!\n";
56        exit(EXIT_FAILURE);
57      }
58    }
59    for (int i=0; i<ns; ++i)
60    {
61      if (!not_correct(m[i]))
62      {
63        m[i].print(tf);
64      }
65    }
66    delete[] m;
67
68    if (ff) fin.close();
69    if (tf) fout.close();
70  }
```