Курс лекций "Программирование"
Основы программирования на языках C и C++

Лекция 8. Процедурное и модульное программирование средствами языков C и C++

Глухих Михаил Игоревич, к.т.н., доц.

mailto: glukhikh@mail.ru

Разбиение программы на модули (файлы)

- Самые простые программы могут состоять из одной функции main
- Чуть более сложные включают в себя другие функции
- По мере возрастания сложности программы функций становится слишком много, в них становится тяжело ориентироваться
- Выход – разбиение функций на отдельные модули по смысловому значению

Пример

- Во входном файле приведены координаты точек на плоскости. Необходимо составить из этих точек три треугольника максимальной площади и вывести их в выходной файл

Какие модули можно выделить в данной задаче?

- Есть следующие группы задач:
  - ввод-вывод (модуль **file**)
  - геометрия (модуль **geometry**)
  - поиск треугольников наибольшей площади (модуль **search**)

Как организуется модуль на языке C/C++?

- Модуль логически состоит из двух файлов - файла с исходным кодом (**source file**) и заголовочного файла (**header file**)
  - файл с исходным кодом (**module.cpp**) включает в себя определения функций, а также определения глобальных переменных и констант (если они есть); в первой строчке обычно подключается заголовочный файл того же модуля:

    ```
    #include "module.h"
    ```
  - заголовочный файл (**module.h**) включает в себя прототипы функций, определения констант, объявления глобальных переменных - но только для тех элементов модуля, о которых должны знать другие модули

Как организуется модуль на языке C/C++?

- Если какой-либо модуль использует данный, необходимо подключить его заголовочный файл (в двойных кавычках – это указывает на то, что заголовочный файл не системный, а собственный):

```
#include "module.h"
```

- Главный модуль программы обычно содержит только функцию **main** и не имеет заголовочного файла (поскольку функция **main** не используется в других модулях)
- Некоторые модули включают только заголовочный файл – например, содержащий определения глобальных констант

Что такое «объявление переменной»?

- Запись типа
  ```
  extern int Var;
  ```
- Которая указывает, что в одном из файлов с исходным кодом на верхнем уровне определена глобальная переменная:
  ```
  int Var;
  ```
- Объявленную переменную можно использовать, даже если в данном файле с исходным кодом нет ее определения
- Если несколько модулей используют одни и те же константы, они обычно определяются в заголовочном файле; объявления констант не используются

Как происходит сборка программы?

- Сборка (build) состоит из 3 основных этапов:
  - на этапе **препроцессинга (preprocessing)** директивы препроцессора (например, **#include**) заменяются содержимым указанного в них заголовочного файла, в результате файл с исходным кодом дополняется прототипами указанных там функций и объявлениями глобальных переменных – за счет этого в файле можно вызывать указанные функции (прототип впереди) и использовать глобальные переменные (объявление впереди). Препроцессор может создавать на диске временные файлы, которые, однако, удаляются после окончания сборки

Как происходит сборка программы?

- Сборка (build) состоит из 3 основных этапов:
  - на этапе **компиляции (compiling)** для каждого исходного файла составляются таблицы определенных в нем функций и глобальных переменных, все определенные функции переводятся на машинный язык (при переводе на машинный язык могут выявляться ошибки компиляции). Результатом работы компилятора являются файлы **module.obj** (объектные файлы) для каждого использованного в программе модуля

Как происходит сборка программы?

- Сборка (build) состоит из 3 основных этапов:
  - на этапе **связывания (linking)** происходит привязка всех используемых (вызванных) функций и глобальных переменных к той таблице, в которой они определены; если определения не обнаружены ни в одной таблице, происходит ошибка связывания (unresolved external symbol …). Результатом связывания является файл **program.exe** (для MVS имя совпадает с именем проекта) – исполняемый файл

Что при сборке происходит с библиотечными функциями?

- Прототипы функций находятся в заголовочных файлах (iostream, math.h, stdio.h, string.h и т.п.)
- Объектные файлы с уже переведенными на машинный язык определениями функций заранее собраны в библиотеки (файлы с расширением lib или dll)
  - определения из статических библиотек (расширение lib) на этапе связывания добавляются в исполняемый файл программы
  - определения из динамических библиотек (расширение dll) в исполняемый файл не добавляются; вместо этого в ссылке на соответствующую функцию указывается, что она находится в динамической библиотеке, и при ее вызове происходит обращение к библиотеке
  - поэтому, статические библиотеки нужны только на этапе связывания, а динамические – и на этапе исполнения программы

Пример. Заголовочный файл geometry.h

```
#ifndef _GEOMETRY_H
#define _GEOMETRY_H

// Расстояние между двумя точками по их координатам
double calcDistance(double xa, double ya, double xb,
   double yb);

// Площадь треугольника по координатам точек
double calcAreaByPoints(double xa, double ya, double
   xb, double yb, double xc, double yc);

#endif
```

Что такое #ifndef-#define-#endif?

- Это, как и **#include**, директивы препроцессора
- Директива **#define** позволяет определить переменную препроцессора (например, _GEOMETRY_H)
- Директива **#ifndef** позволяет выяснить, определена ли переменная препроцессора. Если она **не определена**, участок кода до директивы **#endif** вставляется в программу, если же **определена** – выбрасывается из нее
- Нужны эти скобки для того, чтобы текст заголовочного файла не мог вставиться в исходный файл дважды

Как лучше хранить в памяти набор точек на плоскости

- Каждая точка имеет x-координату и y-координату. Поэтому нам необходимо сохранить два массива: px[] и py[].
- Лучше было бы создать собственный тип Point (точка) и один массив point[], но этого мы пока не умеем.

Как лучше хранить в памяти набор треугольников

- Каждый из треугольников имеет три точки, у каждой из которых две координаты. Поэтому можно создать шесть массивов: ax[], ay[], bx[], by[], cx[], cy[]
  - такой подход возможен, но неудобен – слишком много переменных
- Предлагается создать двумерный массив с x-координатами трех точек: tx[][3] и y-координатами трех точек: ty[][3].
  - число переменных сокращается до двух
- А идеалом было бы создать собственный тип Triangle и хранить массив triangles[].
  - пока не умеем

Пример. Заголовочный файл file.h

```
#ifndef _FILE_H
#define _FILE_H

// Подсчет числа точек во входном файле
int countPoints(const char* fileName);
// Чтение точек из входного файла
bool readPoints(const char* fileName, double* px,
   double* py, int maxPointNum);
// Вывод треугольников в выходной файл
bool writeTriangles(const char* fileName,
   const double tx[][3], const double ty[][3],
   int trNum);

#endif
```

Пример. Заголовочный файл search.h

```
#ifndef _SEARCH_H
#define _SEARCH_H

// Поиск треугольников максимальной площади
void searchLargestTriangles(const double* px,
   const double* py, int pointNum, double tx[][3],
   double ty[][3], int maxTrNum);

#endif
```

Теперь попробуем написать функцию main

- Имена входного и выходного файлов будем читать из командной строки
- Нам необходимо
  - прочитать имена файлов и открыть их
  - посчитать точки во входном файле
  - записать их в массив
  - найти наибольшие треугольники
  - вывести результат в выходной файл

Пример. Главный файл main.cpp

```
#include "file.h"
#include "search.h"
#include <iostream>
#include <locale.h>
using namespace std;
int main(int argc, char** argv)
{
   setlocale(LC_ALL, "Russian");
   if (argc < 2)
   {
      cout<<"Запуск: Triangles.exe inf.txt outf.txt"<<endl;
      return -1;
   }
   const char* inFileName = argv[1];  // Имя входного файла
   const char* outFileName = argv[2]; // Имя выходного файла
   int pointNum = countPoints(inFileName);
```

Пример. Главный файл main.cpp

```
if (pointNum < 0)
{
   cout<<"Входной файл не существует"<<endl;
   return -2;
} else if (pointNum < 4)
{
   cout<<"Входной файл слишком мал"<<endl;
   return -3;
}
double* px = new double[pointNum]; // точки
double* py = new double[pointNum]; // точки
if (!readPoints(inFileName, px, py, pointNum))
{
   cout<<"Ошибка при вводе точек из файла"<<endl;
   return -3;
}
```

Пример. Главный файл main.cpp

```
    const int maxTrNum = 3;
    // Треугольники
    double trX[maxTrNum][3], trY[maxTrNum][3];
    searchLargestTriangles(px, py, pointNum,
       trX, trY, maxTrNum);
    if (!writeTriangles(outFileName, trX, trY, maxTrNum))
    {
       cout<<"Не удалось записать результат в файл"<<endl;
       return -4;
    }
    cout<<"Программа успешно завершена"<<endl;
    delete[] px;
    delete[] py;
    return 0;
}
```

Геометрические функции

- Расстояние между точками:
  - $((x_2-x_1)^2+(y_2-y_1)^2)^{0.5}$
- Площадь треугольника – используем формулу Герона:
  - полупериметр: p=(a+b+c)/2
  - площадь: $s=(p*(p-a)*(p-b)*(p-c))^{0.5}$

Пример. Файл geometry.cpp

```
#include "geometry.h"
#include <math.h>
double calcDistance(double xa, double ya,
   double xb, double yb)
{
   return sqrt((xb-xa)*(xb-xa)+(yb-ya)*(yb-ya));
}
double calcAreaBySides(double a, double b, double c)
{
   double p2=(a+b+c)/2.0;
   // Не забываем проверить, что корень извлечется
   if (p2<=0.0 || p2<=a || p2<=b || p2<=c)
      return 0.0;
   return sqrt(p2*(p2-a)*(p2-b)*(p2-c));
}
```

Пример. Файл geometry.cpp

```
double calcAreaByPoints(double xa, double ya,
   double xb, double yb, double xc, double yc)
{
   double ab=calcDistance(xa, ya, xb, yb);
   double bc=calcDistance(xb, yb, xc, yc);
   double ca=calcDistance(xc, yc, xa, ya);
   return calcAreaBySides(ab, bc, ca);
}
```

Обратите внимание

- Функции имеют много однотипных аргументов. При вызове легко что-нибудь перепутать. Вообще говоря, это плохо.
- Уменьшить количество аргументов можно, вводя собственные типы.

Файл file.cpp, функция countPoints

```
// Подключить file.h, fstream, std
int countPoints(const char* fileName)
{
   ifstream in(fileName);
   if (!in.is_open()) return -1;
   double x,y;
   int pointNum=0;
   int i;
   for (i=0; ; i++)
   {
      in>>x>>y;
      if (in.fail()) break;
   }
   return pointNum;
}
```

Файл file.cpp, функция readPoints

```
bool readPoints(const char* fileName,
   double* px, double* py, int maxPointNum)
{
   ifstream in(fileName);
   // Файл не открыт
   if (!in.is_open())
      return false;
   for (int i=0; i<maxPointNum; i++)
   {
      in>>px[i]>>py[i];
      if (in.fail())
         return false;
   }
   return true;
}
```

Файл file.cpp, функция writeTriangles

```
bool writeTriangles(const char* fileName,
   const double tx[][3], const double ty[][3], int trNum)
{
   ofstream out(fileName);
   if (!out.is_open())
      return false;
   for (int i=0; i<trNum; i++)
   {
      out<<"#"<<i+1<<": A=("<<tx[i][0]<<","<<ty[i][0]<<
                        ") B=("<<tx[i][1]<<","<<ty[i][1]<<
                        ") C=("<<tx[i][2]<<","<<ty[i][2]<<
                        ") S="<<calcAreaByPoints(
                        tx[i][0], ty[i][0], tx[i][1],
                        ty[i][1], tx[i][2], ty[i][2])<<endl;
   }
   return true;
}
```

Как искать треугольники наибольшей площади?

- Надо перебрать все тройки точек (например, в тройном цикле)
- Следует завести массив, в котором будут храниться наибольшие площади по убыванию. Изначально заполним его нулями (реальные площади гарантированно больше)
- Когда определена площадь очередного треугольника, следует определить, нужно ли ее вставить в массив (функция **findPosition**)
- И, если нужно, необходимо «раздвинуть» массив, убрав крайний элемент, и на освободившееся место вставить очередную площадь. Ту же операцию необходимо выполнить с массивами точек треугольников (функции **insert**)

Файл search.cpp, функция findPosition

```
#include "search.h"
#include "geometry.h"

// Массив arr[] упорядочен по убыванию и имеет длину length
// Ищем, куда в него вставить значение value
// Результат от 0 до length
int findPosition(double* arr, int length, double value)
{
   int pos;
   for (pos=length-1; pos>=0; pos--)
   {
      if (value <= arr[pos])
         break;
   }
   return pos+1;
}
```

Файл search.cpp, функции insertElement

```
// Массив arr[] имеет длину length
// Вставить в него элемент value по индексу pos,
// сдвинув остальные элементы вправо (и вытолкнув последний)
void insertElement(double* arr, int length,
   double value, int pos)
{
   for (int m=length-2; m>=pos; m--)
      arr[m+1]=arr[m];
   arr[pos]=value;
}
```

Файл search.cpp, функции insertElement

```
// Имя функции то же, а аргументы другие
// Массив arr[][3] имеет длину length
// Вставить в него элемент value по индексу arr[pos][npoint],
// сдвинув остальные элементы вправо (и вытолкнув последний)
void insertElement(double arr[][3], int length,
   double value, int pos, int npoint)
{
   for (int m=length-2; m>=pos; m--)
      arr[m+1][npoint]=arr[m][npoint];
   arr[pos][npoint]=value;
}
```

Файл search.cpp, функция searchLargestTriangles

```
void searchLargestTriangles(const double* px,
   const double* py, int pointNum,
   double tx[][3], double ty[][3], int maxTrNum)
{
   if (maxTrNum <= 0)
      return;
   double* maxTrArea = new double[maxTrNum];
   for (int i=0; i<maxTrNum; i++)
   {
      maxTrArea[i] = 0.0;
      for (int j=0; j<3; j++)
      {
         tx[i][j] = 0.0;
         ty[i][j] = 0.0;
      }
   }
```

Файл search.cpp, функция searchLargestTriangles

```
   for (int i=0; i<pointNum; i++) // Перебор троек точек
      for (int j=i+1; j<pointNum; j++)
         for (int k=j+1; k<pointNum; k++)
         {
            // ...
         }
   delete[] maxTrArea;
}
```

Файл search.cpp, функция searchLargestTriangles

```
// Тело тройного цикла
double currArea = calcAreaByPoints(px[i], py[i],
   px[j], py[j], px[k], py[k]);
int numInMax = findPosition(
   maxTrArea, maxTrNum, currArea);
if (numInMax==maxTrNum)
   continue;
insertElement(maxTrArea, maxTrNum,
   currArea, numInMax);
insertElement(tx, maxTrNum, px[i], numInMax, 0);
insertElement(tx, maxTrNum, px[j], numInMax, 1);
insertElement(tx, maxTrNum, px[k], numInMax, 2);
insertElement(ty, maxTrNum, py[i], numInMax, 0);
insertElement(ty, maxTrNum, py[j], numInMax, 1);
insertElement(ty, maxTrNum, py[k], numInMax, 2);
```

Вот и все!

- Общая длина файлов – около 200 строк
- Главный минус – большое количество аргументов у функций