Автор: Андрей
23.12.2009 18:18 - Обновлено 29.12.2009 16:52

Долог путь наставлений, краток и убедителен путь примеров
Луций Анней Сенека

Предисловие

Итак, на протяжении семи уроков мы с вами овладели минимальным набором знаний и умений для написания простейших игр на C++. Однако "простейших" не значит "скучных"! Инструментов, описанных в предыдущих уроках вполне хватит, чтобы написать игру под названием "пятнашки". На ее примере я покажу, как происходит создание программы на C++, о чем вы должны думать при работе над программой, чем обусловлен выбор того или иного инструмента языка и как вообще из отдельных элементов собрать цельную программу. В книгах и учебниках по программированию почему-то эти вопросы часто игнорируются, внимание уделяется только отдельным элементам языка, поэтому новичку в программировании приходится самому изобретать велосипед, пытаясь понять, как из этого, этого и вот этого создать работающую программу. Поэтому данный урок будет посвящен скорее общим идеям программирования, в частности - программирования на C++.

Быть может, кое-что из нижесказанного покажется вам описанным излишне подробно, а что-то вы и вовсе сочтете излишним, однако я рекомендую вам очень внимательно отнестись к моим рассуждениям и советам - они содержат действительно нужные и полезные сведения, которые пригодятся вам при написании и более серьезных программ.

Итак, приступим!

Шаг первый: постановка задачи

Самый первый вопрос, который вы должны себе задать, звучит примерно так: "Что должна делать эта программа? Какие функции она должна выполнять? Какими особенностями должна она обладать? Как должна выглядеть работа программы?" Ответы на эти вопросы составят приблизительный список требований, предъявляемый к программе и помогут нам яснее представить себе, что именно нам предстоит сделать.

В нашем случае я представил себе ответы на эти вопросы (и, соответственно,

реализацию игры) так: "Программа должна реализовывать игру "Пятнашки". Первоначально поле должно быть перемешано произвольным образом, а игрок должен перемещать костяшки до тех пор, пока поле не будет собрана должным образом. При этом задача должна быть обязательно разрешимой. Рисование будет производиться с помощью псевдографики, а управление должно осуществляться с помощью клавиатуры." Пока это все, что нам нужно. Исходя из такой постановки задачи можно уже представить себе примерную структуру программы.

Шаг второй: проектирование структуры программы

Следующий не менее вопрос, который вам предстоит себе задать, после того, как вы определились с конкретизацией своей цели: "Какова должна быть структура программы? Из каких частей она должна состоять? Как эти части должны взаимодействовать друг с другом?" Поверьте мне, не стоит лепить код, если вы даже близко себе не представляете, как будет организована ваша программа. В лучшем случае наделаете кучу лишнего, в худшем - просто не сможете увязать друг с другом отдельные обрывки кода. При увеличении объема проекта важность проектирования возрастает в геометрической (приблизительно!;)) прогрессии!

На этом этапе нам уже необходимо обратиться к особенностям конкретного языка программирования (вообще говоря, выбор средств и инструментов, таких как конкретный язык программирования предшествует данному этапу, но для нас такой вопрос не стоит, поэтому мы его опустим). Во-первых, как театр начинается с вешалки, так и выполнение программы, написанной на языке C++, начинается с функции main. Она должна стать центром, в котором происходит обращение ко всем прочим частям и элементам программы. Кроме того, функция main должна во многом отражать структуру программы. Чаще всего в главной функции, и в главном модуле программы содержится меньшая часть кода по сравнению со всеми прочими частями. Итак, в функцию main мы сведем все прочие элементы программы, придав им четкую и ясную структуру. Но сначала определимся с этими самыми элементами.

Работая на языке C++ мы для разделения кода данной конкретной игры применим функции, каждая из которых будет выполнять строго определенные действия, реализуя вполне определенные части поставленной задачи. Итак, как должна выглядеть игра "Пятнашки", и какими данными должна оперировать программа?

Разумеется, что центральным объектом в данной игре является игровое поле размером 4 на 4 клетки. Естественным образом будет представить его в виде массива (краткий ликбез по массивам вы найдете чуть ниже по тексту, а более полный рассказ - в следующих уроках) целых чисел - каждый элемент массива будет содержать в себе номер костяшки. Представляется, что других фундаментальных объектов нам не понадобится.

Теперь представим себе, какие действия и в каком порядке должна будет выполнять программа. Во-первых, нам нужно будет сгенерировать первоначальное расположение костяшек (помним, что мы требуем от этого начального расположения разрешимость всей задачи!). Во-вторых, нам, несомненно, потребуется рисовать поле. В-третьих, игра должна завершиться как только поле будет собрано правильно, поэтому нам понадобится функция, выполняющая проверку правильности текущего положения

костяшек. Сам же процесс игры будет происходить следующим образом. Сначала программа генерирует поле с учетом всех требований. Затем вплоть до завершения сборки поля происходит игровой цикл, включающий в себя: считывание с клавиатуры нажатия клавиши, перемещение костяшек в соответствии с нажатой клавишей, перерисовку поля. И наконец (после выхода из игрового цикла, т.е. после сборки поля) программа поздравляет победителя и завершается.

И здесь мы обнаруживаем, что забыли про функцию, перемещающую костяшки! Это естественно, т.к. прикидывая в уме список функций еще до определения четкой структуры программы мы далеко не всегда можем себе представить, что нам действительно понадобится, а что - нет. К счастью, наша забывчивость не смертельна, и ошибка была обнаружена еще на раннем этапе проектирования, и ничего не мешает нам добавить в список четвертую функцию. Впрочем, в больших проектах подобные (а также более серьезные) ошибки могут произойти позже, что в итоге может привести к кардинальному пересмотру всей структуры программы. Именно поэтому так важно заранее спроектировать программу, чтобы заранее предусмотреть все возможные сложные места и потенциальные ошибки.

Теперь, однако, мы можем вполне четко представить себе структуру программы:

```
    Генерация поля
Игровой цикл (выполняется пока поле не будет собрано):
  Считывание клавиши
  Передвижение костяшек в зависимости от нажатой клавиши
  Рисование поля
Поздравление игрока
```

Это будет уже нетрудно реализовать с помощью средств языка и вызова четырех наших функций, к написанию которых мы теперь и перейдем.

Шаг третий: объявляем переменные и пишем функции

Надеюсь, вам еще не наскучило столь продолжительное словоблудие?;) Мужайтесь, мы переходим непосредственно к программированию! Но не забывайте также, что все, что было сказано выше - действительно важно и представляет вам вполне подходящий образец рассуждений, помогающих писать понятные и эффективные программы!

Итак, начинаем оформлять все эти мысли в виде кода, понятного нашему компилятору. Первым делом мы объявим объект, который будет представлением игрового поля. Как я уже говорил выше, мы представим его в виде массива, причем двумерного, целых чисел. Для каждого типа T существует тип T[N] - массив элементов типа T. Массив представляет собой набор из N элементов (N должна быть константой) данного типа, пронумерованных от 0 до N - 1. Массив объявляется следующим образом:

Теперь, однако, мы можем вполне четко представить себе структуру программы:

```
char str[20]; //Массив из 20 символов  const int M = 5;  std::string Book[M]; //Массив из 5 станда
```

К i-му элементу массива Type Array[Size] мы обратимся с помощью конструкции Array[i]:

```
      double series[1000];
 double sum = 0;
 for (int i = 0; i                  {
   series[i] = 1 / i;
   sum += series[i];
   }
```

Многомерные массивы представляются как массивы массивов: int Matrix[3][3] - массив элементов типа массив целых чисел, по сути - двумерный массив (матрица) целых чисел. Пожалуй, это все, что нам пока надо знать о массивах для написания этой программы.
Итак, мы объявим двумерный массив целых чисел, причем сделаем это в самом начале кода программы, вне блоков, заключенных в фигурные скобки, в том числе - и вне каких-либо функций. Т.е. мы объявим его в *глобальной области видимости*, сделав этот массив *глобальной переменной*. Это облегчит нам жизнь, упростив доступ к нему функций - его не надо будет передавать им в качестве аргумента. Встречайте, первая строка кода нашей программы:

```
int Field[4][4];
```

В дальнейшем нам также удобно будет знать координаты пустой ячейки поля, на которой нет костяшек, и на которую их можно двигать. Поэтому мы также объявим такие две переменные:

```
int EmptyX, EmptyY;
```

Теперь же перейдем к функциям. Самой первой мы напишем функцию генерации поля, коль скоро она у нас первой и вызывается. Определимся сначала с алгоритмом, который мы будем использовать. Как известно, среди всех возможных комбинаций расположения костяшек не всякое является разрешимым. Мы же изначально требовали, чтобы при генерации было получено поле, которое можно привести к правильному виду. Зачастую эту проблему решают не слишком изящно, просто устанавливая сначала поле в правильное положение, а потом перемешивая костяшки по правилам игры. Чтобы получить пристойный результат, надо совершить достаточно большое количество случайных перемещений костяшек, да и алгоритм получается не самым простым.
Но - открою вам секрет! - на самом деле так мучиться не надо! Существует гораздо более простой и рациональный способ определить разрешимое начальное положение костяшек. Для этого мы прибегнем к помощи математической теории игры "пятнашки". Лично я необходимые для этого знания почерпнул давно - после того, как в детском саду стал обладателем книжек "Занимательные задачи для маленьких" и "Смекалка для малышей" (о том, что детям полезно читать правильные книжки даже в наш цифровой

век - как-нибудь в другой раз!;)). Именно, первая из них, составленная на основе книг Я. И. Перельмана, рассказывала об истории этой игры, разрешимых и неразрешимых задачах в ней, а также о способе, позволяющем узнать, можно ли данное расположение костяшек привести к исходному. Суть его заключается в следующем. Пусть у нас имеется некоторое расположение костяшек на поле, причем в правом нижнем углу костяшек нет. *Беспорядком* называется такое положение костяшки, при котором она стоит ранее другой костяшки, имеющей меньший номер. Сколько костяшек с меньшими номерами стоят после данной - таково количество беспорядков для нее. Если общее число беспорядков на всем поле - четное, то его можно привести в правильное расположение, если же нечетное - то этого никак нельзя сделать согласна правилам игры.

Вооружившись этой ценной информацией, приступим теперь к написанию кода функции. Первым делом мы просто-напросто сгенерируем абсолютно случайное расположение костяшек.

```
  void CreateField()
{
bool NumIsFree[15]; //NumIsFree[i] показывает, определили ли мы уже позицию i-й костяшки
int Nums[15]; //Nums[i] содержит номер костяшки, находящейся в i-й позиции
for (int i = 0; i                    NumIsFree[i] = true;
randomize(); //randomize позволяет при каждом прогоне программы получать разные последо
bool Ok; //Флаг, определяющий корректность выбора костяшки для данной позиции
int RandNum; //Номер костяшки, генерируемый в дальнейшем случайным образом
for (int i = 0; i                    {
   Ok = false; //Каждый раз сбрасываем значение флага
   while (!Ok) //Продолжаем случайным образом определять номер костяшки, пока он не окаж
      {
      RandNum = random(15) + 1; //random(n) генерирует псевдослучайное число от 0 до n - 1, а
      if (NumIsFree[RandNum - 1]) //Если костяшка с таким номером еще свободна (помним, что
         Ok = true; //то мы определили ее номер корректно
      }
   Nums[i] = RandNum; //Записываем этот корректный номер в i-ю позицию
   NumIsFree[RandNum - 1] = false; //Костяшка с этим номером теперь занята
   }
}
```

Затем нам нужно будет посчитать общее число беспорядков на поле, и если их окажется нечетное число, то мы поменяем местами костяшки на 14-й и 15-й позиции - при этом число беспорядков изменится на единицу и станет четным. В код функции добавится следующее:

```
     int Chaos = 0; //Количество беспорядков на поле
int CurrNum; //Костяшка, для которой мы рассматриваем беспорядки
for (int i = 0; i                    {
   CurrNum = Nums[i];
```

```
    for (int j = i + 1; j              if (CurrNum > Nums[j])
          Chaos++;
    }
  if (Chaos % 2 == 1) //Если общее число беспорядков нечетное,
    { //меняем местами костяшки на 14-й и 15-й позициях
    int temp = Nums[13];
    Nums[13] = Nums[14];
    Nums[14] = temp;
    }
```

И наконец нам осталось только перевести информацию о расположении костяшек из вида, удобного для анализа, в вид, удобный для игры, а также назначить координаты пустой ячейки:

```
      for (int i = 0; i                Field[i % 4][i / 4] = Nums[i]; //a % b - остаток от деления a на b
  Field[3][3] = 0;
  EmptyX = 3; EmptyY = 3;
```

Следующая функция, которую мы напишем - функция рисования игрового поля. Рисовать мы его будем псевдографически, впрочем, для простоты будем использовать не собственно символы псевдографики, а вполне обычные печатные символы. Каждая костяшка будет занимать пространство четыре символа в ширину и три в высоту, номер будет выводиться посередине второго ряда. Допустим, что "рамку" костяшки мы будем рисовать с помощью знаков +. Т.е. в собранном виде поле будет иметь такой вид:

```
    ++++++++++++++++
+ 1++ 2++ 3++ 4+
++++++++++++++++
++++++++++++++++
+ 5++ 6++ 7++ 8+
++++++++++++++++
++++++++++++++++
+ 9++10++11++12+
++++++++++++++++
++++++++++++
+13++14++15+
++++++++++++
```

Реализовать функцию можно двумя способами: либо рисовать по отдельности каждую костяшку с использованием функции gotoxy, либо рисовать по очереди все 12 рядов символов, представляющих изображение поля с использованием просто перевода строки. Первый способ более логичный и наглядный, однако, плохо переносимый, т.к.

функция gotoxy, устанавливающая курсор в точке с указанными координатами, есть не во всех средах разработки (например, в IDE от фирмы Borland - Borland C++ и Borland C++ Builder она есть, а в Microsoft Visual C++ ее нет). Второй способ менее прозрачный, но использует только стандартные функции. Итак, приведем пример обоих реализаций функции:

```
  void DrawField() //С помощью gotoxy
{
clrscr(); //Функция, очищающая экран
for (int i = 0; i                    for (int j = 0; j                    if (Field[i][j]) //Если в этой позиции есть костя
        {
        gotoxy(i * 4 + 1, j * 3 + 1); //Рисуем первый ряд костяшки
        std::cout
```