基于 Linux 的 C++

第五讲　程序组织与软件开发方法

提　纲

- 库与接口
- 随机数库
- 作用域与生存期
- 典型软件开发流程

库与接口

库与程序文件

程序文件：源文件（*.cpp）、头文件（*.h、*.hpp、*）

库：源文件与头文件

接　口

通过接口使用库：包括指定库的头文件与源文件

优势：不需了解库的实现细节，只需了解库的使用方法

标准库

C标准库

标准输入输出库、工具与辅助函数库、字符串库

C++标准库

输入输出流库、字符串库、标准模板库

数学库

数学库

- 头文件：math.h/cmath
- 库文件：libm
- 链接方式：g++ -lm main.cpp

数学函数

- 三角函数与反三角函数系列
- 幂函数与对数函数系列
- 其他数学函数

标准辅助函数库

工具与辅助函数

头文件：stdlib.h/cstdlib

常用函数

```
void exit( int status );
void free( void * p );
void * malloc( size_t size );
int rand();
void srand( unsigned int seed );
```

随机数库

随机数的生成

库的设计原则

随机数库接口

随机数库实现

随机数库测试

随机数的生成第一版

编写程序，调用 rand 函数生成五个随机数

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int i;
  cout << "On this computer, the RAND_MAX is" << RAND_MAX << ".\n";
  cout << "Five numbers the rand function generates as follows:\n";
  for( i = 0; i < 5; i++ )
    cout << rand() << "; ";
  cout << endl;
  return 0;
}
```

随机数的生成第二版

编写程序，调用 rand 函数生成五个随机数

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main()
{
  int i;
  cout << "On this computer, the RAND_MAX is " << RAND_MAX << ".\n";
  cout << "Five numbers the rand function generates as follows:\n";
  srand( (int)time(0) );
  for( i = 0; i < 5; i++ )
    cout << rand() << "; ";
  cout << "\n";
  return 0;
}
```

接口设计原则

用途一致

接口中所有函数都属于同一类问题

操作简单

函数调用方便，最大限度隐藏操作细节

功能充足

满足不同潜在用户的需要

性能稳定

经过严格测试，不存在程序缺陷

随机数库接口

设计随机数接口

```
void Randomize();

int GenerateRandomNumber( int low, int high );

double GenerateRandomReal( double low, double high );
```

随机数库实现

实现随机数库

```
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "random.h"

using namespace std;

void Randomize()
{
  srand( (int)time(NULL) );
}
```

随机数库实现

```
int GenerateRandomNumber( int low, int high )
{
  double _d;
  if( low > high )
  {
    cout << "GenerateRandomNumber: Make sure low <= high.\n";
    exit( 1 );
  }
  _d = (double)rand() / ((double)RAND_MAX + 1.0);
  return (low + (int)(_d * (high - low + 1)));
}
double GenerateRandomReal( double low, double high )
{
  double _d;
  if( low > high )
  {
    cout << "GenerateRandomReal: Make sure low <= high.\n";
    exit( 2 );
  }
  _d = (double)rand() / (double)RAND_MAX;
  return (low + _d * (high - low));
}
```

随机数库测试

单独测试库的所有函数

- 合法参数时返回结果是否正确
- 非法参数时返回结果是否正确，即容错功能是否正常

联合测试

- 多次运行程序，查看生成的数据是否随机
- 测试整数与浮点数随机数是否均能正确工作

作用域与生存期

量的作用域与可见性

量的存储类与生存期

函数的作用域与生存期

声明与定义

量的作用域与可见性

作用域与可见性

作用域：标识符的有效范围

可见性：程序中某个位置是否可以使用某个标识符

标识符仅在其作用域内可见

位于作用域内的标识符不一定可见

局部数据对象

定义于函数或复合语句块内部的数据对象（包括变量、常量与函数形式参数等）

局部数据对象具有块作用域，仅在定义它的块内有效

有效性从定义处开始直到该块结束

多个函数定义同名的数据对象是允许的，**原因？**

局部数据对象的作用域

```
int func( int x, int y )
{
  int t;
  t = x + y;
  /* 单独出现的花括号对用于引入嵌套块 */
  {
    /* 允许在块中定义数据对象，作用域仅限本块 */
    int n = 2;
    cout << "n = " << n << endl;
  }
  return t;
}
```

量的作用域与可见性

全局数据对象

定义于函数或复合语句块之外的数据对象

全局数据对象具有文件（全局）作用域，有效性从定义处开始直到本文件结束，其后函数都可直接使用

若包含全局数据对象定义的文件被其他文件包含，则其作用域扩展到宿主文件中，**这可能会导致问题，为什么？**

不要在头文件中定义全局数据对象

函数原型作用域

定义在函数原型中的参数具有函数原型作用域，其有效性仅延续到此函数原型结束

函数原型中参数名称可以与函数实现中的不同，也可以省略

作用域与可见性示例

```
1   int i;                                          /* 全局变量 i 作用域开始，可见 */
2   int func( int x );
3   int main()
4   {
5     int n;                                        /* 局部变量 n 作用域开始，可见 */
6     i = 10;                                       /* 全局变量 i 有效且可见 */
7     cout << "i = " << setw(2) << i << "; n = " << n << endl;
8     n = func( i );
9     cout << "i = " << setw(2) << i << "; n = " << n << endl;
10  }                                               /* 局部变量 n 作用域结束，不再可见 */
11  int n;                                          /* 全局变量 n 作用域开始，可见 */
12  int func( int x )                               /* 形式参数 x 作用域开始，可见 */
13  {
14    i = 0;                                        /* 全局变量 i 有效且可见 */
15    cout << "i = " << setw(2) << i << "; n = " << n << endl;
16    n = 20;                                       /* 全局变量 n 有效且可见 */
17    {                                             /* 嵌套块开始 */
18      int i = n + x; /* 局部变量 i、x 有效可见；全局变量 n 有效可见；全局变量 i 有效不可见 */
19     cout << "i = " << setw(2) << i << "; n = " << n << endl;
20    }                                             /* 局部变量 i 作用域结束，全局变量 i 有效且可见 */
21    return ++i;
22  }                                               /* 局部变量 x 作用域结束，不再可见 */
23                                                  /* 文件结束，全局变量 i、n 作用域结束 */
```

量的存储类与生存期

生存期：量在程序中存在的时间范围

- C/C++ 使用存储类表示生存期
- 作用域表达量的空间特性，存储类表达量的时间特性

静态（全局）生存期

- 全局数据对象具有静态（全局）生存期
- 生死仅与程序是否执行有关

自动（局部）生存期

- 局部数据对象具有自动（局部）生存期
- 生死仅与程序流程是否位于该块中有关
- 程序每次进入该块时就为该对象分配内存，退出该块时释放内存；
- **两次进入该块时使用的不是同一个数据对象**

static 关键字

修饰局部变量：静态局部变量

使局部变量具有静态生存期

程序退出该块时局部变量仍存在，并且下次进入该块时使用上一次的数据值

静态局部变量必须进行初始化

不改变量的作用域，仍具有块作用域，即只能在该块中访问，其他代码段不可见

修饰全局变量

使其作用域仅限定于本文件内部，其他文件不可见

静态局部变量示例

检查下述程序的输出结果

```
#include <iostream>
using namespace std;
int func( int x );
int main()
{
  int i;
  for( i = 1; i < 4; i++ )
    cout << "Invoke func " << i << " time(s): Return " <<  func(i) << ".\n";
  return 0;
}
int func( int x )
{
  static int count = 0;  /* 定义静态局部变量count，函数结束后仍存在 */
  cout << "x = " << x << ".\n";
  return ++count;
}
```

函数的作用域与生存期

所有函数具有文件作用域与静态生存期

在程序每次执行时都存在，并且可以在函数原型或函数定义之后的任意位置调用

内部函数与外部函数

外部函数：可以被其他文件中的函数所调用

内部函数：不可以被其他文件中的函数所调用

函数缺省时均为外部函数

内部函数定义：使用 static 关键字

内部函数示例：static int Transform(int x);

内部函数示例：static int Transform(int x){ … }

声明与定义

声明不是定义

定义在程序产生一个新实体

声明仅仅在程序中引入一个实体

函数的声明与定义

声明是给出函数原型，定义是给出函数实现代码

类型的声明与定义

产生新类型就是定义

类型定义示例：typedef enum __BOOL { FALSE, TRUE } BOOL;

不产生新类型就不是定义，而仅仅是声明

类型声明示例： enum __BOOL;

全局变量的作用域扩展

全局变量的定义不能出现在头文件中，只有其声明才可以出现在头文件中

声明格式：使用 extern 关键字

```
/* 库的头文件 */
/* 此处仅引入变量 a，其定义位于对应源文件中 */
extern int a;  /* 变量 a 可导出，其他文件可用 */

/* 库的源文件 */
/* 定义变量 a */
int a;
```

典型软件开发流程

- 软件工程概要
- 问题的提出
- 需求分析
- 概要设计
- 详细设计
- 编码实现
- 系统测试
- 经验总结

软件工程概要

需求分析：确定软件需要解决什么问题

决定因素：人

- 软件开发人员需要与用户深入交流，明确问题的输入、输出以及其他附加信息
- 不要轻视任何问题

方案设计：设计程序框架

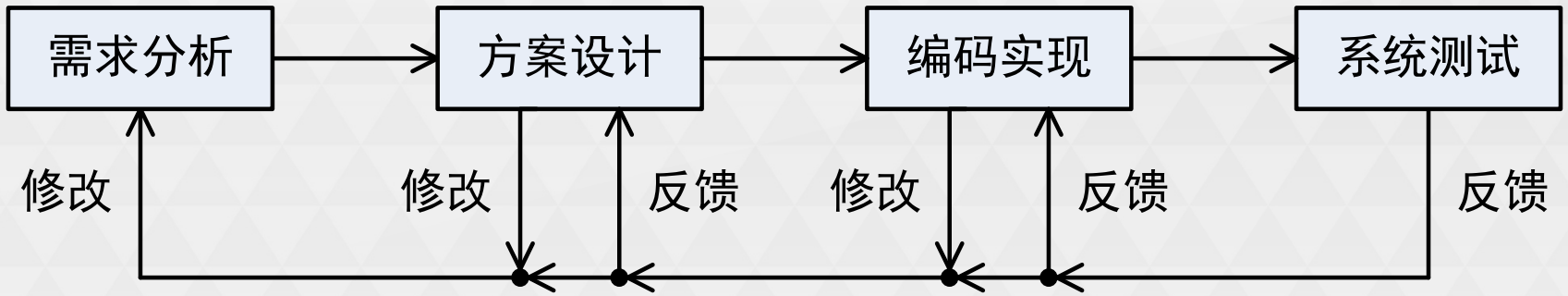
- 概要设计：设计总体方案，形成高层模块划分
- 详细设计：细化模块，获得各模块的输入、输出与算法

编码实现：实际编程

系统测试：测试程序的正确性与稳定性

经验总结

软件开发流程图



软件开发问题

我猜！我猜！我猜猜猜！

编程实现一个简单的猜价格游戏

假设有某物品，已知其最低价格与最高价格

游戏参与者在给定次数内猜测其价格具体值

需求分析

需求A：游戏运行前首先应向游戏参与者介绍游戏功能

需求B：首期工程不需要解决游戏难度问题，用户迫切希望程序能在最短时间内运行起来，因此只考虑最简单情形

需求C：在每个游戏回合结束时允许用户选择是否重新开始新游戏，这里游戏回合是指游戏参与者或者猜中价格或者其猜测机会已用完，如果用户没有选择退出，游戏应无休止地玩下去

需求D：能够记录游戏参与者的游戏信息，目前仅统计用户玩了多少回合以及赢了多少回合

需求E：在用户退出游戏时，给出此次游戏胜率

需求分析

不明确的地方：什么是最简单情形？与用户再次沟通……

需求B1：设物品最低价格为100元，最高价格为200元

需求B2：物品实际价格由系统运行时随机生成

需求B3：游戏参与者最多允许猜6次

需求B4：若游戏参与者猜测价格比实际价格高，则程序提示“高”；

若猜测价格比实际价格低，提示“低”

用户补充需求

需求F：需要定义游戏初始化过程，未来有可能通过它调整游戏难度，

程序必须为此提供接口

概要设计

响应需求 A、E、F

将游戏划分为四个模块：欢迎信息显示模块、游戏初始化模块、游戏模块、游戏结束模块

程序架构：主文件“main.cpp”，程序主体函数库“guess.h”与“guess.cpp”；此外需要使用 zyrandom 库

设计函数原型

胜率信息需要从游戏模块传递给游戏结束模块

```
void PrintWelcomeInfo();
void InitializeGame();
double PlayGame();
void PrintGameOverInfo( double prevailed_ratio );
```

编码实现：guess.h

```
void PrintWelcomeInfo();

void InitializeGame();

double PlayGame();

void PrintGameOverInfo( double prevailed_ratio );
```

编码实现：main.cpp

```
#include "guess.h"

int main()
{
  double prevailed_ratio;
  PrintWelcomeInfo();
  InitializeGame();
  prevailed_ratio = PlayGame();
  PrintGameOverInfo( prevailed_ratio );
  return 0;
}
```

概要设计

响应需求 C

使用无限循环作为游戏模块的主架构：初始化游戏回合，进行游戏回合，判断游戏参与者是否开始新游戏回合

响应需求 B2

每一回合需要随机生成物品价格，应在游戏回合初始化阶段完成

抽象出单独的游戏回合初始化函数，返回值为初始化的物品价格（整数类型）

```
int InitializeBout();
```

响应需求 D

游戏参与者的回合数与获胜回合数：使用整数类型保存，游戏开始时初始化为 0，并随着游戏进程变化

概要设计

游戏回合活动分析

将每一回合游戏抽象成函数，函数返回值为 bool，表示该游戏回合游戏参与者是否获胜

```
bool PlayBout();
```

判断用户是否进入下一回合

```
bool Again();
```

Again函数负责在 PlayGame 函数中充当哨兵，在返回值为 false 时退出游戏

响应需求 B1、B3：数据初始化工作

```
const int lowest_price = 100;
const int highest_price = 200;
const int guess_count = 6;
```

详细设计

响应需求 B4：细化 PlayBout

在给定猜测次数内接受游戏参与者的输入价格，判断与实际价格是否相同，如果不同则给出提示信息，如果相同则恭喜游戏参与者获胜

但是……

需求 G：一旦游戏参与者给出了猜测价格，当该价格或高或低时，提示游戏参与者的信息应能够相应缩小价格区间以反映此变化

响应需求 G：修改 PlayBout 函数，添加参数

```
bool PlayBout( int actual_price, int lower_price, int
higher_price, int chances_left );
```

详细设计

欢迎信息模块设计

要求：将游戏性质与游戏方法告诉游戏参与者，系统应尽可能输出详细信息，例如物品最低价格、最高价格与猜测次数都应通知游戏参与者

伪代码

```
void PrintWelcomeInfo()
{
  输出程序性质信息
  物品最低价格、最高价格与猜测次数一并输出
}
```

详细设计

游戏初始化模块设计

要求：不知道

伪代码

```
void InitializeGame()
{

}
```

详细设计

游戏结束模块设计

要求：如果能够在游戏结束模块不仅输出游戏参与者胜率，还针对其胜率高低输出不同鼓励信息就好了

需求 H：当游戏参与者胜率超过75%（含）时，输出“You luckyyyyyyyyyyyyy!”；当胜率低于75% 但超过50%（含）时，输出“So gooooooood.”；其他输出“You can do it better. Wish you luck.”

伪代码

```
void PrintGameOverInfo( double prevailed_ratio )
{
  输出百分制胜率
  根据胜率分别输出不同信息
}
```

详细设计

游戏模块细化

伪代码

```
double PlayGame()
{
  while( true ){
    调用 InitializeBout 获得实际价格
    调用 PlayBout 启动游戏回合
    如果游戏回合结束后结果为 true，递增获胜回合数
    递增总回合数
    调用 Again，若结果为 false，终止游戏，否则继续
  }
  返回最终胜率
}
```

详细设计

游戏回合初始化与游戏初始化模块细化

伪代码

```
int InitializeBout()
{
  调用 GenerateRandomNumber 函数
  并返回其结果
}

void InitializeGame()
{
  调用 Randomize 函数启动随机数发生器
}
```

详细设计

游戏回合模块细化

需求 I：如果游戏参与者给出的猜测价格不在当时价格范围内，应提醒用户重新提供新价格，此次操作不应递减用户猜测次数，以防止用户输入错误，也就是说，需要检查猜测价格的合法性

需求 J：当最后一次机会也未猜中价格时，不需要再输出提示价格高低信息，直接输出用户本回合失败信息，并将实际价格告诉用户

详细设计

```
bool PlayBout( int actual_price, int lower_price, int higher_price, int chances_left ){
  while( 还有猜测机会 ){
    获得游戏参与者猜测价格，检查游戏参与者猜测价格是否在给定范围
    递减游戏参与者猜测机会
    判断猜测价格高低
    switch( 判断结果 ){
    case 高了:
      if( 还有猜测机会 ){  输出价格高了信息，降低最高价格值，缩小猜测区间  }
      else{  输出失败信息与物品实际价格，结束函数，返回本回合失败值  false  }
      break;
    case 低了:
      if( 还有猜测机会 ){  输出价格低了信息，增大最低价格值，缩小猜测区间  }
      else{  输出失败信息与物品实际价格，结束函数，返回本回合失败值 false  }
      break;
    default:
      输出游戏参与者获胜信息，结束函数，返回 true
    }
  }
  程序流程至此，说明游戏参与者本回合已失败，返回 false
}
```

详细设计

进一步细化游戏回合函数

抽象三个辅助函数

获得游戏参与者猜测价格：int GetPrice(int lower_price, int higher_price, int chances_left);

检查游戏参与者猜测价格是否在给定范围：int CheckPrice(int lower_price, int higher_price, int guess_price);

判断猜测价格高低：int JudgePrice(int actual_price, int guess_price);

详细设计

```
int GetPrice( int lower_price, int higher_price, int chances_left )
{
  输出提示信息，包括最低价格，最高价格与剩余猜测次数
  获取猜测价格并返回
}
int CheckPrice( int lower_price, int higher_price, int guess_price )
{
  while(猜测价格小于最低价格或高于最高价格){
    通知游戏参与者猜测价格超出范围，提醒用户重新输入新价格
    获取新猜测价格
  }
  返回新猜测价格
}
int JudgePrice( int actual_price, int guess_price)
{
  获得猜测价格与实际价格之差
  if(差值大于0) return 1; else if(差值小于0) return -1; else return 0;
}
```

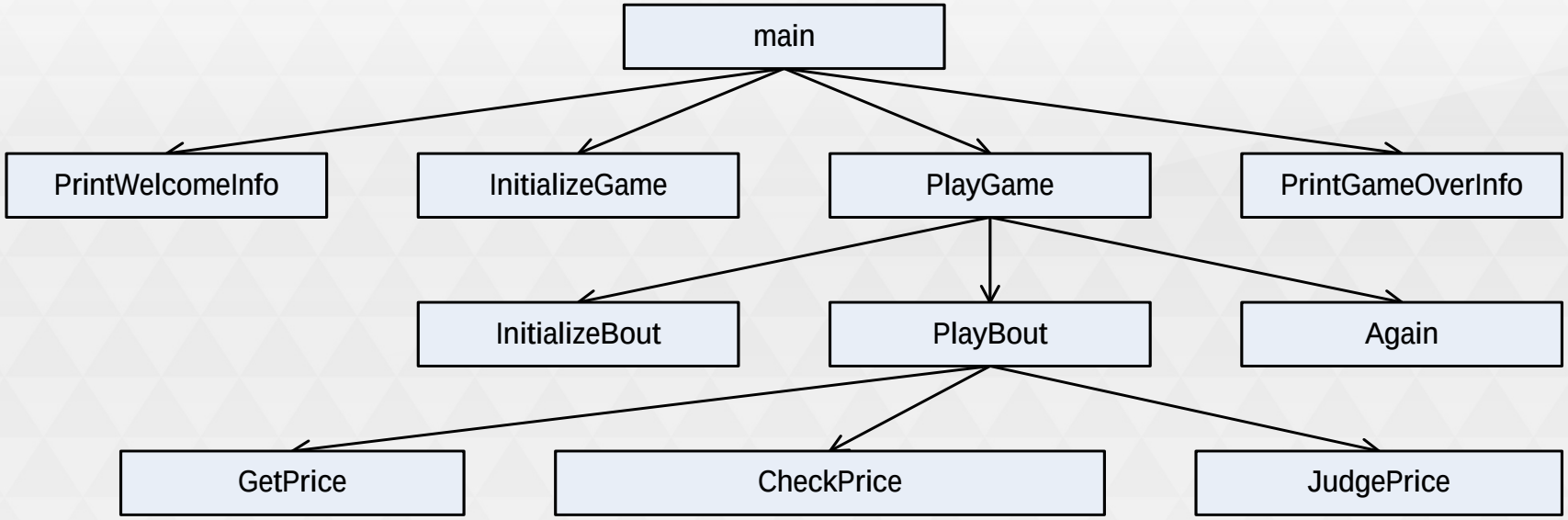
详细设计

细化 Again 函数

伪代码

```
bool Again()
{
  询问游戏参与者是否开始新游戏
  获取游戏参与者的响应
  判断游戏参与者的响应是否为数字 0
  如果真，返回 false，即将上述返回值取反返回
}
```

详细设计


main
PrintWelcomeInfo
InitializeGame
PlayGame
PrintGameOverInfo
InitializeBout
PlayBout
Again
GetPrice
CheckPrice
JudgePrice

编码实现：guess.cpp

```
#include <iostream>
#include <iomanip>

#include "random.h"
#include "guess.h"

using namespace std;

const int lowest_price = 100;
const int highest_price = 200;
const int guess_count = 6;
```

编码实现：guess.cpp

```
static int InitializeBout();

static bool PlayBout(int actual_price, int lower_price, int
  higher_price, int chances_left);

static bool Again();

static int GetPrice(int lower_price, int higher_price, int
  chances_left);

static int CheckPrice(int lower_price, int
  higher_price, int guess_price);

static int JudgePrice(int actual_price, int
  guess_price);
```

编码实现：guess.cpp

```
void PrintWelcomeInfo()
{
  cout << "The program lists a product with price between "
    <<  lowest_price << " and " << highest_price << " (RMB Yuan).\n";
  cout << "You give a guess price. If the price you give is "
    << "correct, you win.\n";
  cout << "You have " << guess_count << " chances.\n";
  cout << "Rise to the challenge to win your bonus...\n";
}

void InitializeGame()
{
  int i, n;
  Randomize();
}
```

编码实现：guess.cpp

```
double PlayGame()
{
  int actual_price, lower_price = lowest_price, higher_price = highest_price;
  int chances_left = guess_count;
  int bout_count = 0, prevailed_bout_count = 0;
  while(true)
  {
    cout << endl;
    actual_price = InitializeBout();
    if(PlayBout(actual_price, lower_price, higher_price, chances_left))
      prevailed_bout_count++;
    bout_count++;
    if(!Again())
      break;
  }
  return (double)prevailed_bout_count / (double)bout_count;
}
```

编码实现：guess.cpp

```
void PrintGameOverInfo(double prevailed_ratio)
{
  cout << "\nprevailed ratio: " << setw(3) << prevailed_ratio * 100 << "%.\n";
  if(prevailed_ratio >= 0.75)
    cout << "You luckyyyyyyyyyyyy!\n\n";
  else if(prevailed_ratio >= 0.50)
    cout << "So goooooooood.\n\n";
  else
    cout << "You can do it better. Wish you luck.\n\n";
}

static int InitializeBout()
{
  return GenerateRandomNumber(lowest_price, highest_price);
}
```

编码实现：guess.cpp

```
static bool PlayBout(int actual_price, int lower_price, int higher_price, int chances_left)
{
  int guess_price;
  int judge_result;
  while(chances_left > 0){
    guess_price = GetPrice(lower_price, higher_price, chances_left);
    guess_price = CheckPrice(lower_price, higher_price, guess_price);
    chances_left--;
    judge_result = JudgePrice(actual_price, guess_price);
    switch(judge_result){
    case 1:
      if(chances_left > 0){  cout << "\nHigher.\n";  higher_price = guess_price - 1;   }
      else{  cout << "\nYou lose this bout. The actual price is " << actual_price << endl;  return false;  }
      break;
    case -1:
      if(chances_left > 0){  cout << "\nLower.\n";  lower_price = guess_price + 1;  }
      else{  cout << "\nYou lose this bout. The actual price is " << actual_price << endl;  return false;  }
      break;
    default:   cout << "\nYou winnnnnnnnnnnnnnn!\n";  return true;
    }
  }
  return false;
}
```

编码实现：guess.cpp

```
static bool Again()
{
  int t;
  cout << "\nPlay a new game (\"0\" to stop, other numbers to play again)? ";
  cin >> t;
  return t != 0;
}

static int GetPrice( int lower_price, int higher_price, int chances_left )
{
  int t;
  cout << "The actual price is in [" << lower_price << ", " << higher_price << "]. ";
  cout << "Chances left " << chances_left << ".\nYou guess: ";
  cin >> t;
  return t;
}
```

编码实现：guess.cpp

```
static int CheckPrice(  int lower_price, int higher_price, int guess_price )
{
  int t = guess_price;
  while( t < lower_price || t > higher_price )
  {
    cout << "Guess price " << t << " is out of the range. ";
    cout << "Please choose one in [" << lower_price << ", " << higher_price
      << "].\nTry again: ";
    cin >> t;
  }
  return t;
}

static int JudgePrice( int actual_price, int guess_price )
{
  int t = guess_price - actual_price;
  if(t > 0)  return 1;  else if(t < 0)  return -1;  else  return 0;
}
```

系统测试

所有数据对象是否已正确初始化？

随机生成的价格是否合理有效？测试过程可能需要在源程序中添加必要的测试代码，例如：

```
double PlayGame()
{
    ……
    actual_price = InitializeBout();
#ifndef NDEBUG
    cout <<"Debugging: Actual price = " << actual_price << endl;
#endif
    ……
}
```

系统测试

检查游戏回合运行是否准确

参数传递是否准确？

获取的猜测价格是否与游戏参与者的实际输入吻合？

价格有效性检查是否正确工作？分别输入准确值、高于最高价格值与低于最低价格值，查看程序运行结果

价格判断是否正确运行？

猜测次数的控制与显示是否准确反映了用户猜测过程的变化？

提示信息是否准确显示？

在显示未猜中信息时，是否最后一次与其他次不同？

系统测试

判断开始新游戏的过程是否准确

检查胜率计算是否准确

检查胜率百分比的计算与显示是否准确

不同鼓励语是否显示准确

- 测试必须覆盖三类可能胜率
- 测试人员试玩游戏，得到三种不同胜率结果

经验总结

Q：这个问题一开始大家认为它难不难？

A：不难。

Q：程序设计难不难？

A：难，如果要程序员从系统分析、方案设计开始做起的话。

Q：编码难不难？

A：不难，与系统分析与设计相比编码实在简单，不过就是使用了所有代码控制结构和函数编写原则而已。

Q：和用户交流烦不烦？

A：烦！

编程实践

5.1 编写函数，返回1～52之间的随机数。

5.2 可以将上题生成的随机数模拟为不含大小王牌的扑克牌。编写函数，重复生成52个随机数，并映射为每张扑克牌。说明：重复生成的典型原则是按照花色（梅花、方块、红桃、黑桃）和大小（2～10、J、Q、K、A）顺序进行映射，例如梅花2小于梅花3，……，梅花A小于方块2，……，黑桃K小于黑桃A。需要注意的是，一旦生成某张牌后，即不允许再次生成它，如何解决此问题？