CS 1502H 程序设计思想与方法 (C++)

Chapter 9. 模块化设计

陈东尧
上海交通大学
2025年秋季

回顾

- 结构体主要用于处理更复杂的数据
- 声明结构类型时，不产生内存的分配
- 定义结构变量时，才会分配内存空间
- 作为函数参数传递时，宜采用引用传递

```
struct studentLinkT {
    char  no[14];
    char  name[12];
    int math;
    int english;
    int programming;
};
```

返回结构体**类型**的函数

- 一个函数返回一个结构体，如：

```
studentT getStudentData( )
{
    studentT student;
    ……
    return(student);
}
```

- 返回的是一个结构体的复制(拷贝)

- 调用函数中可以有这样的程序段

```
int main()
{
    studentT s1;
    s1 = getStudentData();
     return 0;
}
```

返回结构体**引用**的函数

- 函数返回一个结构体的引用。如：

```
studentT  &getStudentData()
{
    studentT  *student  =  new  studentT;
    …….
    return(*student);
}
```

- 函数中返回的结构体不能是局部变量

- 调用函数中可以有这样的程序段

```
int main()
{
    studentT &s1 = getStudentData();
    ……

    //不需要的时候，需要回收变量空间
    delete &s1;
}
```

Quiz：选择题

对结构体变量stu1中成员age的非法引用是（ ）

```
struct studentT {
    int num;
    float age;
} ;
studentT stu1,  *p;
p=&stu1;
```

A. stu1.age　　B. studentT.age

C. p->age　　D. (*p).age

Quiz：选择题

对结构体变量stu1中成员age的非法引用是（B）

```
struct studentT {
    int num;
    float age;
};
studentT stu1,  *p;
p=&stu1;
```

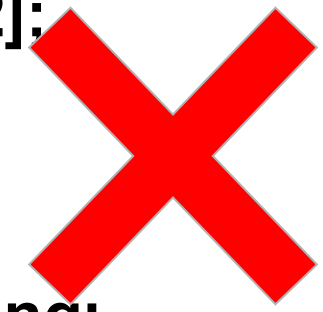
A. stu1.age　　B. studentT.age

C. p->age　　D. (*p).age

扩展

- 我们提到结构体不可嵌套，但可以嵌套结构体指针

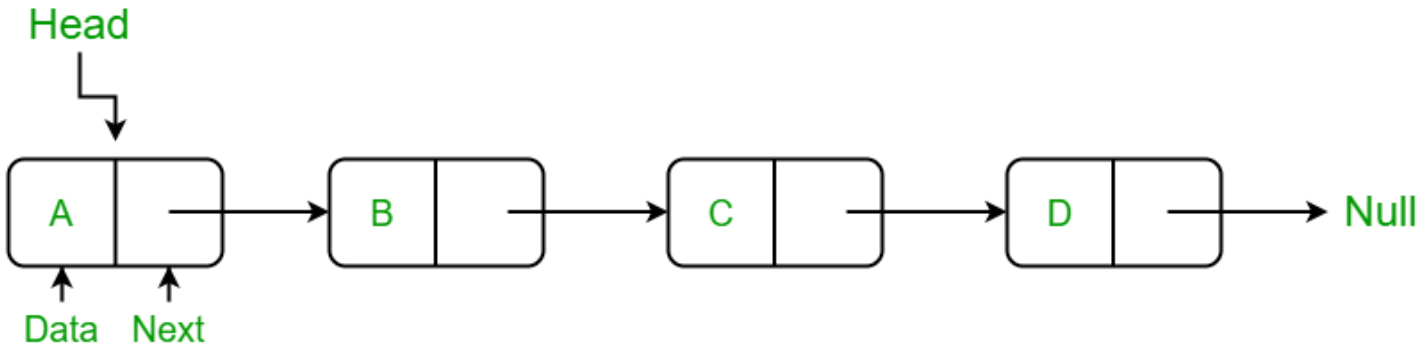
```
struct studentT {
    char  no[14];
    char  name[12];
    int math;
    int english;
    int programming;
    studentT  subStu;
};
```



```
struct studentLinkT {
    char  no[14];
    char  name[12];
    int math;
    int english;
    int programming;
    studentLinkT  *next;
};
```

扩展

- 我们提到结构体不可嵌套，但可以嵌套结构体指针
- 事实上，数据结构中的链表即是这样实现的

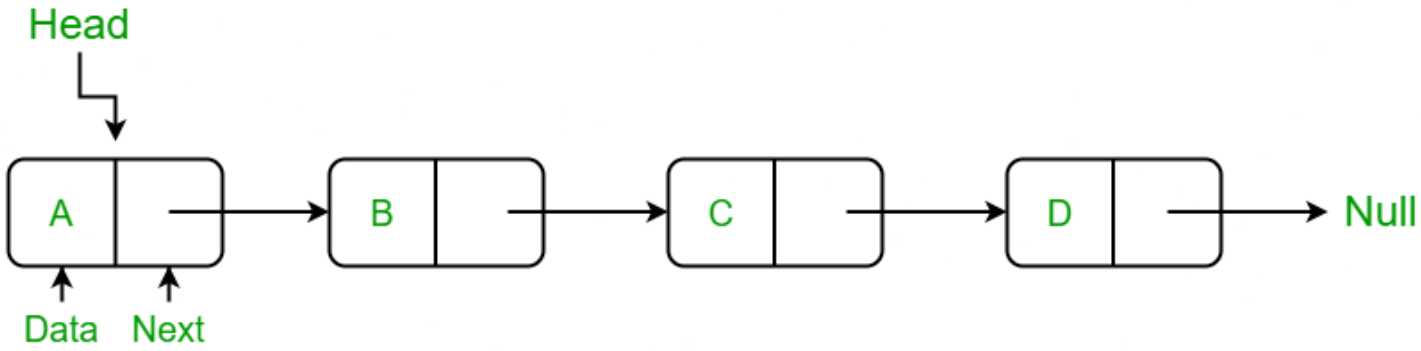



```
// C implementation
struct Node {
    int data;
    Node* next;
};
```

```
//C++ implementation
class Node {
public:
    int data;
    Node* next;
};
```

扩展

- 我们提到结构体不可嵌套，但可以嵌套结构体指针
- 事实上，数据结构中的链表即是这样实现的




```
// C implementation
struct Node {
    int data;
    struct Node* next;
};
```

```
//C++ implementation
class Node {
public:
    int data;
    Node* next;
};
```

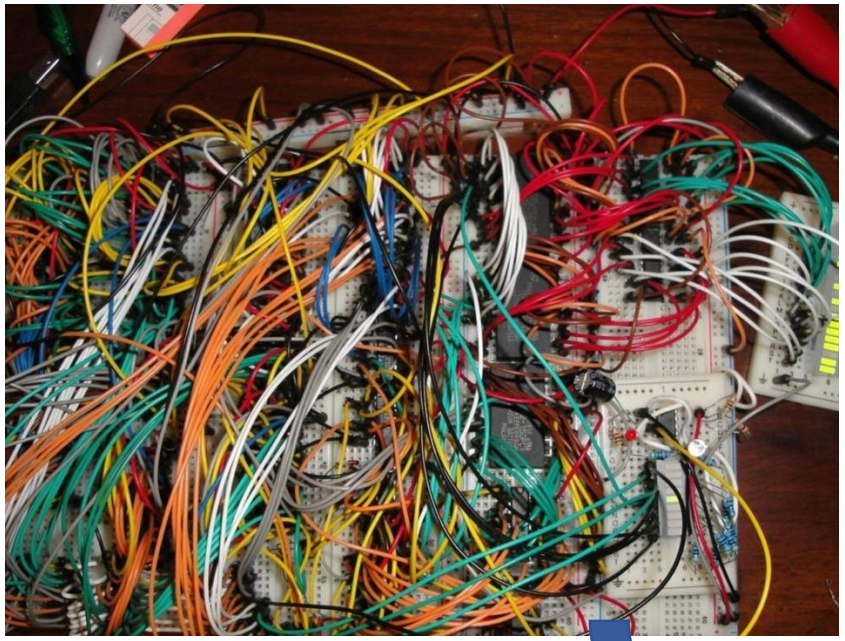
第十章（类）见！

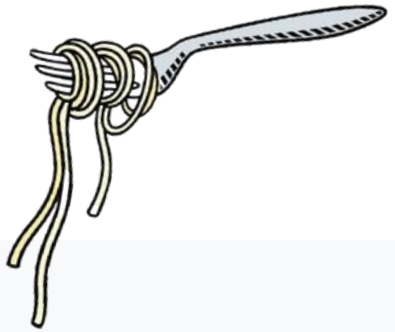
第9章 模块化开发

- **为什么要模块化**
- 模块划分
- 库的设计与实现
- 库的应用

模块化程序设计

- 一个反面教材：Spaghetti Code
  曾经的BASIC语言

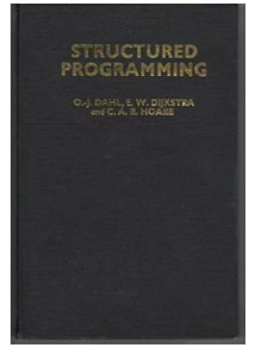




```
1 i=0;
2 i=i+1;
3 PRINT i; "squared=";i*i;
4 IF i>=100 THEN GOTO 6;
5 GOTO 2;
6 PRINT "Program Completed.";
7 END
```

“GOTO statement considered harmful”

Edsger Dijkstra
（迪杰斯特拉）




```
1 FOR i=1 TO 100
2     PRINT i;"squared=";i*i
3 NEXT i
4 PRINT "Program Completed."
5 END
```

模块化设计

模块化程序设计

分工和协作让不可能成为可能

如何解决复杂的大问题？

- 若想要解决这样的问题，一定要
  - 找到一个线头
  - 耐心拆开
  - 重复这一过程

结构化程序设计

设计阶段

自顶向下的分解

每个小问题设计为一个函数

公共小问题可以设计成一个库

实现阶段

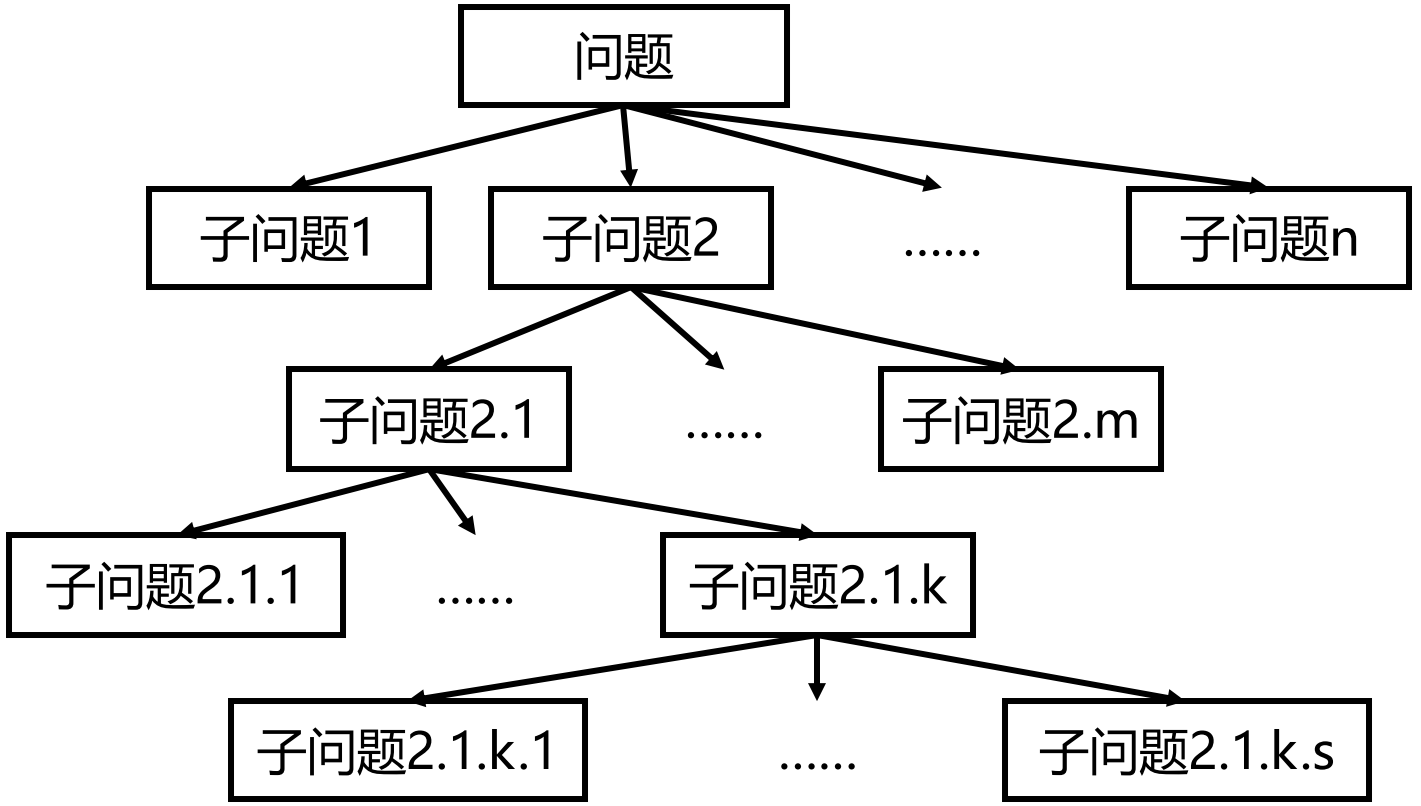
自底向上的实现

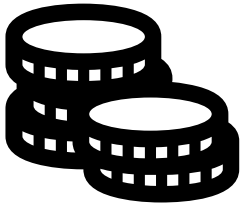
程序由3种基本结构组成

顺序

分支

循环





猜硬币的游戏

功能

提供游戏指南

计算机随机产生正反面，让用户猜，报告对错结果

重复玩游戏过程，直到用户不想玩了为止

顶层分解

程序要做两件事：显示程序指南；模拟玩游戏的过程。

顶层分解

程序要做两件事：显示程序指南；模拟玩游戏的过程。

```
main( )
{
    显示游戏介绍;
    玩游戏;
}
```

主程序的两个步骤是相互独立的两个，没有什么联系，因此可设计成两个函数：

void game_instruction()

void play()

```
int main()
{
    game_instruction();
    play();

    return 0;
}
```

game_instruction的实现

game_instruction函数的实现非常简单，只要一系列的输出语句把游戏指南显示一下

```
void game_instruction()
{
    cout << "这是一个猜硬币正反面的游戏.\n";
    cout << "我会扔一个硬币，你来猜.\n";
    cout << "如果猜对了，你赢，否则我赢。\n";
}
```

play函数的实现

Play函数随机产生正反面，让用户猜，报告对错结果，然后询问是否要继续玩

```
void play()
{
        char flag =  'y' ;
        while ( flag ==  'Y'  || flag ==  'y' )  {
                coin = 生成正反面;
                输入用户的猜测;
                if  (用户猜测 == coin)
                        报告本次猜测结果正确;
                else
                        报告本次猜测结果错误;
                询问是否继续游戏
        }
}
```

生成正反面

正反面表示：用0表示正面，1表示反面

生成正反面：生成0和1两个随机数

输入用户的猜测

直接用一个输入语句

但想让程序做得好一点，就必须考虑得全面一些

比如，用户既不输入0也不输入1

解决方案：抽象成一个函数get_call_from_user。

实现随机数的生成

生成接近完美的随机数是非常有挑战的，人们甚至会用熔岩灯（lava lamp）的像素信息作为随机数种子

play函数的实现

```
void play()
{
    int coin ;
    char flag = 'Y';

    srand(time(NULL));          //设置随机数种子
    while (flag == 'Y' || flag =='y') {
        coin = rand() * 2 / (RAND_MAX+1); //生成扔硬币的结果
        if  (get_call_from_user() == coin)
                cout << "你赢了";
        else
                cout << "我赢了";
        cout << "\n继续玩吗（Y或y）？ ";
        cin >> flag;
   }
}
```

get_call_from_user的实现

该函数接收用户输入的一个整型数。如果输入的数不是0或1，则重新输入，否则返回输入的值

```
int get_call_from_user()
{
	int  guess;		// 0 = head, 1 = tail
	do {
		cout << "\n输入你的选择（0表示正面，1表示反面）:";
		cin >> guess;
	} while (guess  !=0 && guess !=1) ;

	return guess;
}
```

运行实例

这是一个猜硬币正反面的游戏.

我会扔一个硬币，你来猜.

如果猜对了，你赢，否则我赢。

输入你的选择（0表示正面，1表示反面）:1

我赢了

继续玩吗（Y或y）？ y

输入你的选择（0表示正面，1表示反面）:6

输入你的选择（0表示正面，1表示反面）:1

你赢了

继续玩吗（Y或y）？ n

Press any key to continue

第9章 模块化开发

- 为什么要模块化
- **模块划分**
- 库的设计与实现
- 库的应用

模块划分

当程序变得更长的时候，要在一个单独的源文件中处理如此众多的函数会变得困难

解决方案

把程序再分成几个小的源文件。每个源文件都包含一组相关的函数

一个源文件被称为一个模块

模块划分标准

同一模块中的函数比较类似

块内联系尽可能大，块间联系尽可能小

模块划分：例子

paper, rock, scissor (p_r_s)

游戏规则：

布覆盖石头，石头砸坏剪刀，剪刀剪碎布

游戏过程：

游戏者选择出石头、剪子或布，计算机也随机选择一个，输出结果，继续游戏，直到游戏者选择结束为止。在此过程中，游戏者也可以阅读游戏指南或看看当前战况

第一层的功能分解

```
While (用户的选择 != quit) {
  switch(用户的选择){
        case paper, rock, scissor:   机器选择;
                                     评判结果;
                                     报告结果;
        case game: 显示目前的战况;
        case help: 显示帮助信息;
        default: 报告错误;
        }
}
显示战况;
```

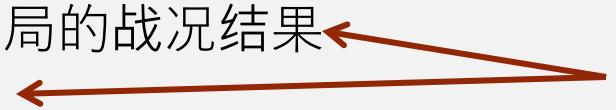
第一层的功能分解

```
While (用户的选择 != quit) {
  switch(用户的选择){
       case paper, rock, scissor:   机器选择;
                                    评判结果;
                                    报告结果;
       case game: 显示目前的战况;
       case help: 显示帮助信息;
       default: 报告错误;
       }
}
显示战况;
```

函数抽取

selection_by_player()：获取用户输入
selection_by_machine()：获取机器输入
compare()：接受用户与机器的输入，并评判结果
report()：报告本次结果，并更新多局的战况结果
prs_game_status()：显示多局的战况结果
prs_help()：显示帮助信息

功能较为相关



数据关系紧密

模块划分

- 主模块**main.cpp**：main()
- 获取选择模块**select.cpp**：selection_by_player()和selection_by_machine()
- 比较模块**compare.cpp**：compare()
- 输出模块**report.cpp**：report(), prs_game_status()和prs_help()

模块划分

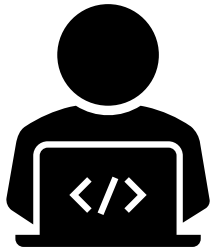
- 主模块**main.cpp**：main()
- 获取选择模块**select.cpp**：selection_by_player()和selection_by_machine()
- 比较模块**compare.cpp**：compare()
- 输出模块**report.cpp**：report(), prs_game_status()和prs_help()

注：每个模块都可由不同程序员编写，可以单独编译

C++中的头文件（header file）

- 头文件中常包涵如下三个关键信息：
  - 函数的定义 （Function definitions）
  - 数据类型的定义 （Data type definitions）
  - 宏 （Macros）

- 调用时用：#include “header.h” or #include <header>



代码演示

calSum.h

calSum.cpp

枚举类型的定义

为了提高程序的可读性，我们定义两个枚举类型：

```
enum p_r_s {paper, rock, scissor, game, help, quit};
enum outcome {win, lose, tie, error};
```

因为各个模块都需要使用这两个枚举类型，我们定义一个头文件 **p_r_s.h**:

```
// 文件: p_r_s.h
// 本文件定义了两个枚举类型

enum p_r_s {paper, rock, scissor, game, help, quit};
enum outcome {win, lose, tie, error};
```

select模块的设计与实现

selection_by_player()从键盘接收用户的输入并返回此输入值。
原型为 p_r_s selection_by_player();

selection_by_machine()由机器产生一个石头、剪子、布的值。
原型为 p_r_s selection_by_machine();

- 为了使用**select**模块，需提供**select**模块中所有函数的原型声明，可以定义**select.h**头文件中：

```
// 文件：select.h

#include "p_r_s.h"
p_r_s selection_by_player();
p_r_s selection_by_machine();
```

select模块的设计

```
//文件: select.cpp
//包括机器选择selection_by_machine和
//玩家选择selection_by_player函数的实现

#include "select.h"
#include <iostream>
#include <cstdlib>
using namespace std;

p_r_s selection_by_machine( )
{ int select = (rand( ) * 3 / (RAND_MAX + 1));

  cout << " I am ";
  switch(select){
      case 0: cout << "paper. "; break;
      case 1: cout << "rock. "; break;
      case 2: cout << "scissor. "; break;
  }
  return ((p_r_s) select);
}
```

select模块的实现

```
p_r_s selection_by_player()
{ char c;
  p_r_s player_choice;

  prs_help(); //显示输入提示
  cout << "please select: "; cin >> c;
  switch(c) {
     case 'p':  player_choice = paper;   cout << "you are paper. "; break;
     case 'r':  player_choice = rock;   cout << "you are rock. "; break;
     case 's':  player_choice = scissor;  cout << "you are scissor.
";break;
     case 'g':  player_choice = game;   break;
     case 'q':  player_choice = quit;  break;
     default : player_choice = help;  break;
  }
  return player_choice;
}
```

compare模块的设计与实现

compare()比较用户输入的值和机器产生的值，确定输赢，需要两个p_r_s类型的参数，返回一个outcome类型的判断的结果。

原型为 outcome compare(p_r_s, p_r_s);

- 定义compare.h:

```
// 文件: compare.h

#include "p_r_s.h"

outcome compare(p_r_s, p_r_s);
```

compare模块的设计与实现

```
//文件：compare.cpp
//包括compare函数的实现

#include "compare.h"
#include <iostream>
using namespace std;

outcome compare(p_r_s player_choice, p_r_s machine_choice)
{ outcome  result;
  if (player_choice == machine_choice)   return tie;
  switch(player_choice) {
     case paper: result = (machine_choice == rock)?win:lose; break;
     case rock: result = (machine_choice == scissor)?win:lose; break;
     case scissor: result =(machine_choice == paper)?win:lose; break;
     default: cout << " PROGRAMMER ERROR: Unexpected choice!\n\n";
     exit(1);  //退出有误，EXIT_FAILURE
    }
  return result;
}
```

print模块的设计与实现

prs_help( )显示一个用户输入的指南，告诉用户如何输入他的选择，不需要参数，也没有返回值。原型为

```
void prs_help();
```

report( )报告输赢结果，并更新总的输赢次数。输的次数、赢的次数、平局的次数、和其他模块的函数无任何关系，因此可作为print模块的内部状态(仅作用于该模块的全局变量)。原型为

```
void report(outcome);
```

prs_game_status( )报告至今为止的战况。原型为

```
void prs_game_status();
```

定义 print.h

```
// 文件: print.h

#include "p_r_s.h"
void prs_help();
void report(outcome);
void prs_game_status();
```

print模块的设计与实现

```
//文件: print.cpp
//包括所有与输出有关的模块。
//有prs_game_status, prs_help和report函数

#include "print.h"

//模块的内部状态
static int win_cnt = 0, lose_cnt = 0, tie_cnt = 0;

void report(outcome result)
{
    switch(result) {
        case win:   ++win_cnt; cout << "You win. \n"; break;
        case lose:  ++lose_cnt; cout << "You lose.\n"; break;
        case tie:   ++tie_cnt; cout <<"A  tie.\n"; break;
        default:
                    cout << " PROGRAMMER ERROR!\n\n";
                    exit(1);
    }
}
```

```
void prs_game_status()
{ cout << endl ;
  cout << "GAME STATUS:" << endl;
  cout << "win:  " << win_cnt << endl;
  cout << "Lose: " << lose_cnt << endl;
  cout << "tie:  " << tie_cnt << endl;
  cout << "Total:" << win_cnt+lose_cnt+tie_cnt << endl;
}

void prs_help()
{ cout << endl
       <<  "The following characters can be used:\n"
       <<  "   p  for paper\n"
       << "   r   for rock\n"
       << "   s   for scissors\n"
       << "   g   print the game status\n"
       << "   h   help, print this list\n"
       << "   q   quit the game\n";
}
```

主模块的实现

```
//文件: main.cpp
// 石头、剪子、布游戏的主模块

#include <iostream>
#include <cstdlib>
#include <ctime>
#include "p_r_s.h"
#include "select.h"
#include "compare.h"
#include "print.h"
using namespace std;

int main(void)
{
   outcome  result;
   p_r_s player_choice, machine_choice;

   // seed the random number generator
   srand(time(NULL));
```

```
    while((player_choice = selection_by_player()) != quit)
       switch(player_choice) {
          case paper:
          case rock:
          case scissor:
               machine_choice = selection_by_machine();
               result = compare(player_choice, machine_choice);
               report(result);  break;
          case game: prs_game_status();  break;
          case help: prs_help();   break;
          default:
             cout<< "PROGRAMMER ERROR!\n\n";
             exit(1);
       }
    prs_game_status(); //退出前输出结果
    return 0;
}
```

注意：头文件的设计

- p_r_s.h会被多次include，造成类型重复定义
- 解决方法：需要用到一个新的编译预处理命令：

```
#ifndef 标识符 //为：“if not defined”
#define 标识符
…
#endif
```

- 重写 p_r_s.h:

```
// 文件：p_r_s.h
// 本文件定义了两个枚举类型

#ifndef  p_r_s_h
#define p_r_s_h
enum p_r_s {paper, rock, scissor, game, help, quit} ;
enum outcome {win, lose, tie, error} ;
#endif
```

其他模块的头文件

- 防止重定义，每个头文件都采用这样的方式定义：

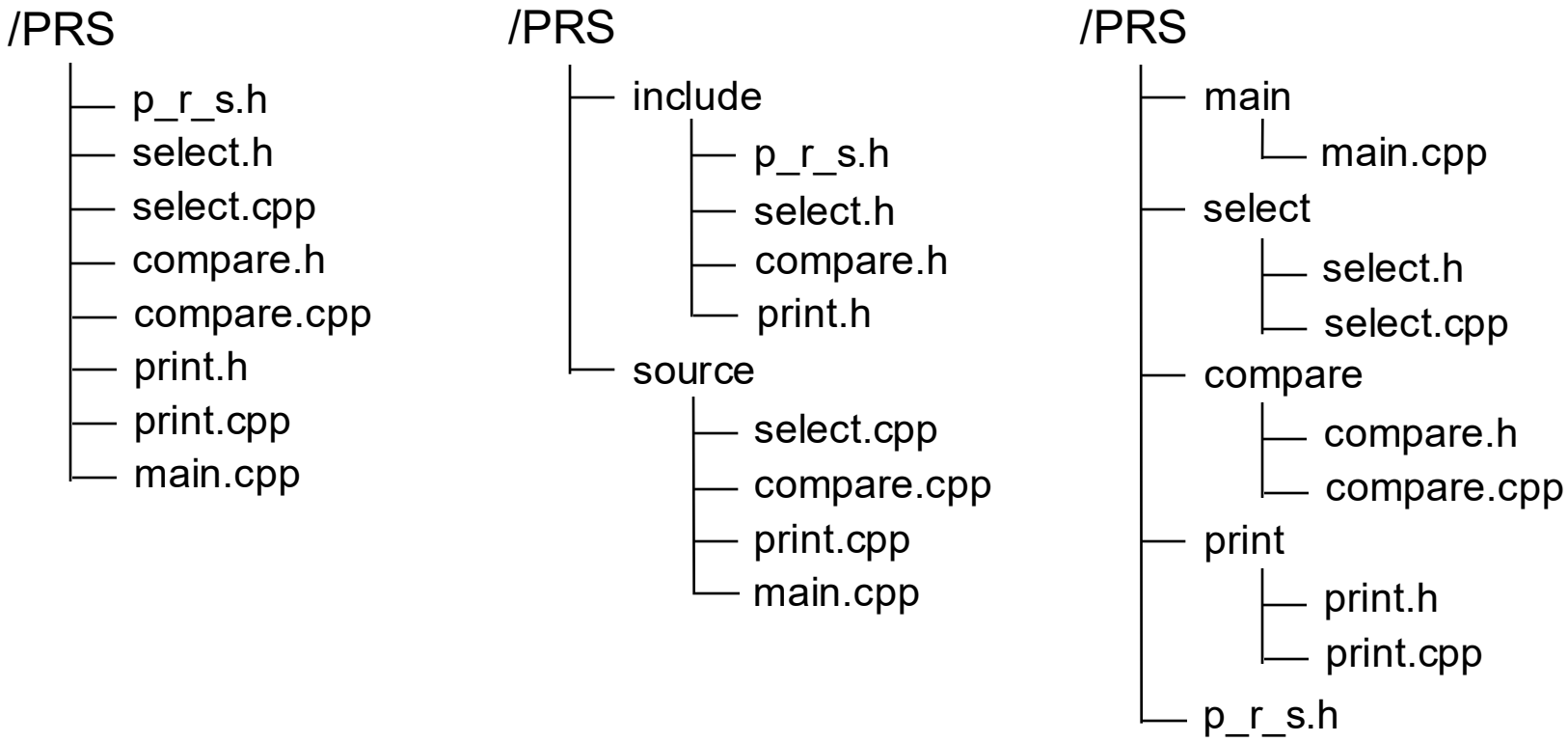
```
// 文件: select.h
#ifndef select_h
#define select_h
#include "p_r_s.h"
p_r_s selection_by_player();
p_r_s selection_by_machine();
#endif
```

```
// 文件: compare.h
#ifndef compare_h
#define compare_h
#include "p_r_s.h"
outcome compare(p_r_s, p_r_s);
#endif
```

```
// 文件: print.h
#ifndef print_h
#define print_h
#include "p_r_s.h"
void prs_help();
void report(outcome);
void prs_game_status();
#endif
```

模块的文件组织与编译方法

- 不同的源文件架构



需要告知编译器：需要哪几个源文件

C++中的编译

- 编译一个程序需要如下三步
  1. 找到文件
  2. 编译
  3. 运行

C++中的编译

- 编译一个程序需要如下三步
  1. 找到文件
  2. 编译
  3. 运行
- 仔细观察我们开发环境（IDE）的输出

```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

C++中的编译

- 命令行控制的方法：

  指令 + 参数
  例如：编译一般是g++或gcc

- 仔细观察我们开发环境（IDE）的输出

```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

其他的是什么?

C++中的编译第一步：找到文件

- cd指令
  - chdir (change directory)

```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

C++中的编译第二步：编译

- 熟悉我们的编译指令

  指令 + 参数
  例如：编译一般是g++或gcc

- 那么， 编译我们的程序的指令一般是

  g++ calSum.cpp -o calSum

- 仔细观察我们开发环境（IDE）的输出

```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

C++中的编译第二步：编译

- 熟悉我们的编译指令

  指令 + 参数
  例如：编译一般是g++或gcc

- 那么， 编译我们的程序的指令一般是

  g++ calSum.cpp -o calSum

  第一个参数：源文件名　　告诉编译器第二个参数的意义　　第二个参数：编译完成的二进制文件名

- 仔细观察我们开发环境（IDE）的输出

```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

C++中的编译第三步: 运行

- 指明要运行哪个文件夹中的哪个文件

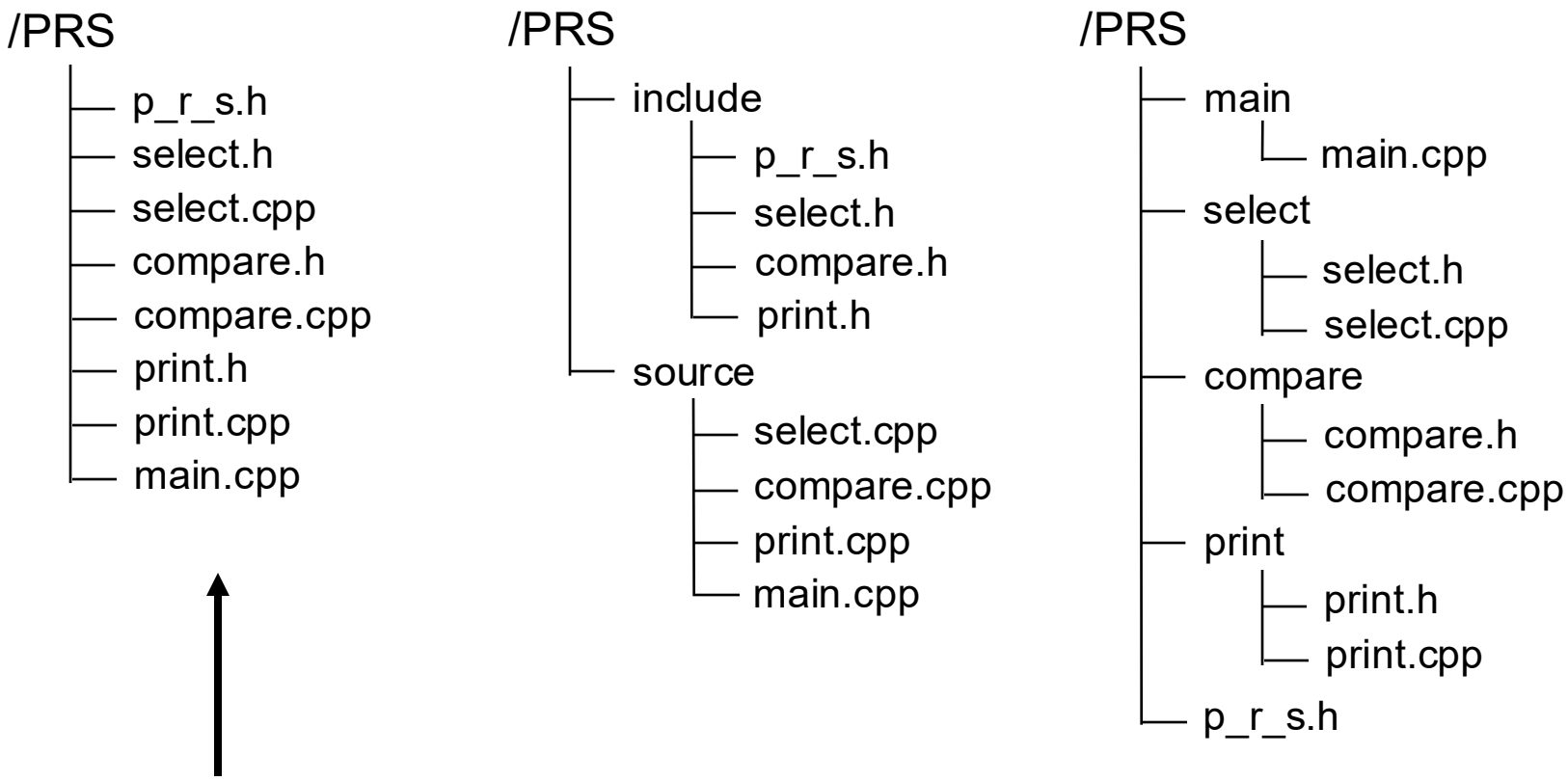
```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

C++中的编译

- 命令行的不同指令可以放在一起，用 &&隔开

```
(base) chendy@ChenDongyao-MacBook-Pro 9-modular_design % cd "/U
sers/chendy/Documents/Teaching/CS1501-2022/
9-modular_design/" && g++ calSum.cpp -o calSum && "/Users/chend
y/Documents/Teaching/CS1501-2022/9-modular_
design/"calSum
```

模块的文件组织与编译方法



对于第一种

g++ select.cpp print.cpp compare.cpp main.cpp -o main

第9章 模块化开发

- 为什么要模块化
- 模块划分
- **库的设计与实现**
- 库的应用

设计自己的库

- 你可以将自己常用的一组函数设计成一个库
  - 如:标准库iostream包含输入输出功能，cmath包含数学运算函数
  - 你可以自己定义库，来方便自己后续的使用
- 设计一个库还要考虑到它的通用性
  - 库函数的功能是从一类相似的应用中抽象概括而来，要考虑尽可能广泛的适用范围

库的设计和实现

- **设计库的接口（interface）：**
  - 库的用户必须了解的内容，包括库中函数的原型、这些函数用到的符号常量和自定义类型
  - 表现为一个**头文件**
- **实现库中的函数（function）：**
  - 表现为一个**源文件**

随机函数库的设计

之前我们设计了一个掷硬币的程序。该程序用到了随机数的一些特性。如果我们的工作经常需要用到随机数，我们可以把随机数的应用写成一个库。

前面的例子，用到了随机生成0和1

随机函数库的设计

之前我们设计了一个掷硬币的程序。该程序用到了随机数的一些特性。如果我们的工作经常需要用到随机数，我们可以把随机数的应用写成一个库。

前面的例子，用到了随机生成0和1
石头剪刀布，用到了随机生成0到2

随机函数库的设计

之前我们设计了一个掷硬币的程序。该程序用到了随机数的一些特性。如果我们的工作经常需要用到随机数，我们可以把随机数的应用写成一个库。

前面的例子，用到了随机生成0和1
石头剪刀布，用到了随机生成0到2
在之前自动出题例子（分支程序设计）中，用到了随机生成0到3及随机生成0到9

随机函数库的设计

之前我们设计了一个掷硬币的程序。该程序用到了随机数的一些特性。如果我们的工作经常需要用到随机数，我们可以把随机数的应用写成一个库。

前面的例子，用到了随机生成0和1
石头剪刀布，用到了随机生成0到2
在之前自动出题例子（分支程序设计）中，用到了随机生成0到3及随机生成0到9

例子：随机数库

- 抽象出一个通用的随机数生成函数：生成low到high之间的随机数
  int RandomInteger(int low, int high) ;
- 初始化函数实现设置随机数种子的功能
  void RandomInit();

库接口的设计

```
//文件：myRandom.h
//随机函数库的头文件

#ifndef myRandom_h
#define myRandom_h

//函数：RandomInit
//用法：RandomInit()
//作用：此函数初始化随机数种子
void RandomInit();

//函数：RandomInteger
//用法：n = RandomInteger(low, high)
//作用：此函数返回一个low到high之间的随机数，包括low和high
int RandomInteger(int low, int high);

#endif
```

库的实现

- 库的实现文件和头文件的名字是相同的。如头文件为myRandom.h，则实现文件为myRandom.cpp
- 实现文件的格式：
  - 注释：这一部分简单介绍库的功能
  - include此cpp文件所需的头文件
  - 每个实现要包含自己的头文件，以便编译器能检查函数定义和函数原型声明的一致性
  - 每个函数的实现代码
  - 在每个函数实现的前面也必须有一段注释

```
//文件：myRandom.cpp
//该文件实现了random库
#include <cstdlib>
#include <ctime>
#include “myRandom.h"

//函数：RandomInit
//该函数取当前系统时间作为随机数发生器的种子
void RandomInit()
{
  srand(time(NULL));
}

// 函数：RandomInteger
// 该函数将随机数转换成low到high区间
int RandomInteger(int low, int high)
{
      return (low + (high-low+1)*rand()/(RAND_MAX+1));
}
```

第9章 模块化开发

- 为什么要模块化
- 模块划分
- 库的设计与实现
- **库的应用**

例子：龟兔赛跑 Tortoise and Hare

动物	移动类型	概率	移动距离
乌龟	快走	50%	前走3点
	后退	20%	后退6点
	慢走	30%	前走一点
兔子	睡觉	20%	不动
	大后退	20%	后退9点
	快跑	10%	前走14点
	小步跳	30%	前走3点
	慢后退	20%	后退2点

总体思路

- 功能
  - 记录龟兔赛跑的移动距离
  - 实现不同行为的概率
- 思路
  - 将时间分段，实现龟兔在每个时间段内的移动行为。
  - 龟兔的移动距离需要分开计算
- 难点
  - 如何实现行为概率？
  - 多模块之间如何整合？

实现随机模块

乌龟	**快走**	**50%**
	后退	**20%**
	慢走	**30%**

可以生成0-9之间的随机数，当生成的随机数为0-4时，认为是第一种情况，5-6是第二种情况，7-9是第三种情况

- 抽象出一个通用的随机数生成函数：生成low到high之间的随机数：
  - int RandomInteger(int low, int high) ;
- 初始化函数实现设置随机数种子的功能 ：
  - void RandomInit();

第一层分解

```
int main()
{
  //timer是计时器，从0开始计时
  int hare = 0, tortoise = 0, timer = 0;

  while (hare < RACE_END && tortoise < RACE_END)
    {    tortoise += 乌龟根据他这一时刻的行为移动的距离;
         hare += 兔子根据他这一时刻的行为移动的距离;
         输出当前计时和兔子乌龟的位置;
         ++timer;
     }
   if (hare > tortoise) cout << "\n hare wins!";
       else cout << "\n tortoise wins!";

return 0;
}
```

龟兔赛跑计数模块

- 乌龟在这一刻的移动距离：

  `int move_tortoise();`

- 兔子在这一刻的移动距离：

  `int move_hare();`

- 输出当前计时和兔子乌龟的位置

  `void print_position(int timer, int tortoise, int hare);`

```
int main()
{ int hare = 0, tortoise = 0, timer = 0;

  RandomInit(); //随机数初始化
  cout << "timer  tortoise  hare\n"; //输出表头
  while (hare < RACE_END && tortoise < RACE_END)
  {   tortoise += move_tortoise(); //乌龟移动
      hare += move_hare(); //兔子移动
      print_position(timer, tortoise, hare);
      ++timer;
   }
   if (hare > tortoise) cout << "\n hare wins!\n";
       else cout << "\n tortoise wins!\n";
   return 0;
}
```

如何用随机数模拟概率

- 以乌龟为例，它的三种情况和概率如下：

快走	**50%**
后退	**20%**
慢走	**30%**

为此我们可以生成0-9之间的随机数，当生成的随机数为0-4时，认为是第一种情况，5-6是第二种情况，7-9是第三种情况。这样就可以根据生成的随机数确定发生的事件

Move模块

```
int move_tortoise()
{
 //产生0到9之间的随机数
 int probability = RandomInteger(0,9);
 if (probability < 5) return 3; //快走
 else if (probability < 7) return -6; //后退
 else return 1; //慢走
}

int move_hare()
{
 int probability = RandomInteger(0,9);
 if (probability < 2) return 0; //睡觉
 else if (probability < 4) return -9; //大后退
 else if (probability < 5) return 14; //快走
 else if (probability < 8) return 3; //小步跳
 else return -2; //慢后退
}
```

Print模块

```
// 文件名：print.cpp

#include <iostream>
using namespace std;

void print_position(int timer, int t, int h)
{
 if (timer % 6 == 0) cout << endl; //每隔6次空一行
     cout << timer << '\t' << t << '\t' << h << '\n';
}
```

总结

- 学会利用模块化程序设计技术来解决一个大问题
- 学会在模块中保存内部状态
- 学会从程序中抽取出库以及如何设计和使用库
- **关键知识点：**
  - 生成随机数（调用什么库函数、以时间为种子、任意范围随机数怎么写）
  - If not defined预防重定义（#ifndef、#define、#endif）
  - 头文件与源文件、联合编译指令