Fundamental Computer Programming- C++ Lab(I)

LAB 9
Craps with functions

Week 10, Fall 2019

Rung-Bin Lin

International Bachelor Program in Informatics
College of Informatics
Yuan Ze University

Functions

- **Function definitions**

```
return-value-type function-name( parameter-list )
{
    declarations and statements
}
```

- **Use of functions to develop structured programs.**
  - **Function prototype (function declaration)**

```
return-value-type function-name( parameter-list );
```

    - ✓ **Name**
    - ✓ **Parameters**
    - ✓ **Return type**
  - **Scope rules (concept of local variables)**
  - **Call a function**

```
function-name( argument-list );
```

  - **Argument coercion, i.e., matching between arguments in function call and parameters in function definition**
  - **return from a function**

return *expression;*

```
1  // Fig. 5.3: fig05_03.cpp
2  // Creating and using a programmer-defined function.
3  #include <iostream>
4  using namespace std;
5
6  int square( int ); // function prototype
7                          // Located outside the main function
8  int main()
9  {
10    // loop 10 times and calculate and output the
11    // square of x each time
12    for ( int x = 1; x <= 10; x++ )
13       cout << square( x ) << "  ";  // function call
14
15    cout << endl;
16 } // end main
17
18 // square function definition returns square of an integer
19 int square( int y )  // y is a copy of argument to function
20 {
21    return y * y;      // returns square of y as an int
22 } // end function square
```

```
1  4  9  16  25  36  49  64  81  100
```

Fig. 5.3 | Programmer-defined function `square`.

Argument Coercion

- **Argument values that do not correspond precisely to the parameter types in the function prototype can be converted by the compiler to the proper type before the function is called.**
  - **For example, `square(4.5)` returns 16, not `20.25`.**
- **These conversions occur as specified by C++'s promotion rules.**

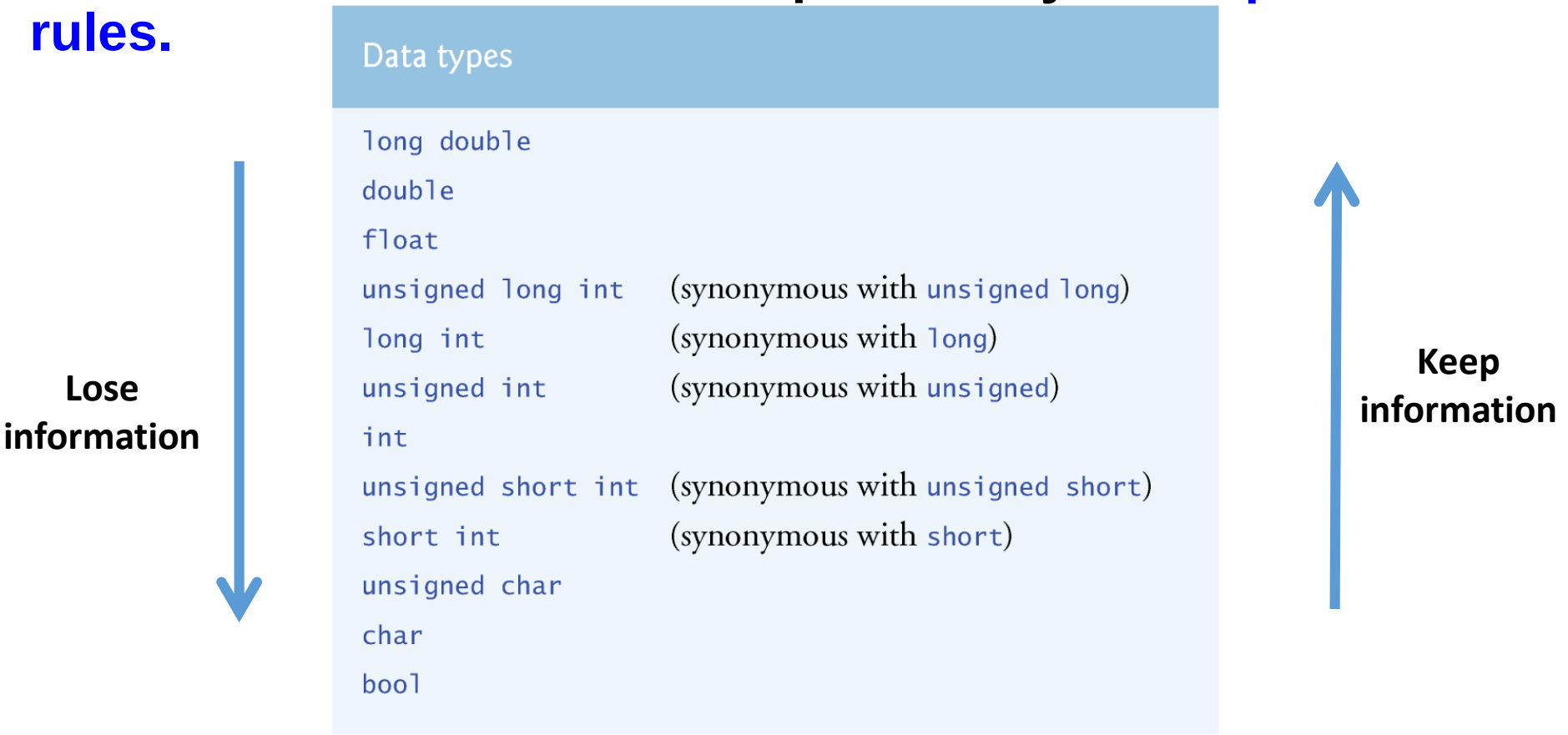

Fig. 5.5 | Promotion hierarchy for fundamental data types.

Examples of Argument Coercion

- **Information lost**

```
Int square(int y)
{
Return y*y;
}

Int main( ) {
double x=2.25;
Cout << square(x) << end;
}
```

- **Information kept**

```
Int square(double y)
{
Return y*y;
}

Int main( ) {
int x=2;
Cout << square(x) << end;
}
```

Enumeration Type

enum Status {CONTINUE, WON, LOST};

- **Status now becomes a type. It can be used to define a variable. For example, Status gameStatus; gameStatus is now a variable of type Status. It can have one of the following three values**
  - ✓ **CONTINUE, assigned a value of 0**
  - ✓ **WON, assigned a value of 1**
  - ✓ **LOST, assigned a value of 2**

rand();

- **A random number generator. The values generated by this function between 0 and RAND_MAX.**

srand(time(0)); // set up a seed

- **time(0) is a function that returns current time in seconds. The return value will be used as a seed for srand(…) function.**

Problem Description of Craps

A player rolls two dice. Each die has six faces. These faces contain 1, 2, 3, 4, 5 and 6 spots. After the dice have come to rest, the sum of the spots on the two upward faces is calculated. If the sum is 7 or 11 on the first roll, the player wins. If the sum is 2, 3 or 12 on the first roll (called "craps"), the player loses (i.e., the "house" wins). If the sum is 4, 5, 6, 8, 9 or 10 on the first roll, then that sum becomes the player's "point." To win, you must continue rolling the dice until you "make your point." The player loses by rolling a 7 before making the point.

The program in Fig. 5.10 simulates the game. In the rules, notice that the player must roll two dice on the first roll and on all subsequent rolls. We define function `rollDice` (lines 63–75) to roll the dice and compute and

Original Code of Craps Game (1)

```
// Fig. 5.10: fig05_10.cpp
// Craps simulation.
#include <iostream>
#include <cstdlib> // contains prototypes for functions srand and rand
#include <ctime> // contains prototype for function time
using namespace std;

int rollDice(); // rolls dice, calculates and displays sum

int main()
{
   // enumeration with constants that represent the game status
   enum Status { CONTINUE, WON, LOST }; // all caps in constants

   int myPoint; // point if no win or loss on first roll
   Status gameStatus; // can contain CONTINUE, WON or LOST
|
```

Original Code of Craps Game (2)

```
// randomize random number generator using current time
srand( time( 0 ) );

int sumOfDice = rollDice(); // first roll of the dice

// determine game status and point (if needed) based on first roll
switch ( sumOfDice )
{
   case 7: // win with 7 on first roll
   case 11: // win with 11 on first roll
      gameStatus = WON;
      break;
   case 2: // lose with 2 on first roll
   case 3: // lose with 3 on first roll
   case 12: // lose with 12 on first roll
      gameStatus = LOST;
      break;
   default: // did not win or lose, so remember point
      gameStatus = CONTINUE; // game is not over
      myPoint = sumOfDice; // remember the point
      cout << "Point is " << myPoint << endl;
      break; // optional at end of switch
} // end switch
```

Original Code of Craps Game (3)

```
// while game is not complete
while ( gameStatus == CONTINUE ) // not WON or LOST
{
   sumOfDice = rollDice(); // roll dice again

   // determine game status
   if ( sumOfDice == myPoint ) // win by making point
      gameStatus = WON;
   else
      if ( sumOfDice == 7 ) // lose by rolling 7 before point
         gameStatus = LOST;
} // end while

// display won or lost message
if ( gameStatus == WON )
   cout << "Player wins" << endl;
else
   cout << "Player loses" << endl;
} // end main
```

Original Code of Craps Game (4)

```
// roll dice, calculate sum and display results
int rollDice()
{
   // pick random die values
   int die1 = 1 + rand() % 6; // first die roll
   int die2 = 1 + rand() % 6; // second die roll

   int sum = die1 + die2; // compute sum of die values

   // display results of this roll
   cout << "Player rolled " << die1 << " + " << die2
      << " = " << sum << endl;
   return sum; // end function rollDice
} // end function rollDice
```

Output of Original Craps Game (5)

```
Player rolled 5 + 1 = 6
Point is 6
Player rolled 5 + 6 = 11
Player rolled 5 + 6 = 11
Player rolled 2 + 5 = 7
Player loses
```

```
Player rolled 6 + 4 = 10
Point is 10
Player rolled 3 + 4 = 7
Player loses
```

```
Player rolled 1 + 3 = 4
Point is 4
Player rolled 1 + 6 = 7
Player loses
```

LAB 9: Craps with functions

- **Modify the program Crap simulation given in Fig. 5.10 in the textbook "C++: How to Program" as follows:**
  - **Rewrite the main function in Fig. 5.10 into a function**
    - ✓ **The function should be crapsFunc().**
    - ✓ **The return value should be a string either "WON" or "LOST".**
    - ✓ **You can modify the game rule to make the player's win probability as close to 0.7 as possible. However, you should follow the style given in the switch statement to determine a WIN or a LOST of each play**
    - ✓ **You should still use rollDice() function to roll dices**
    - ✓ **You should delete lines 55~59 and lines 71~73**
    - ✓ **You should not include line 19 in this function. Include it into the new main() function discussed below.**
  - **Write a new main() function that reads an integer which is the number of times the Craps game will be played. The win probability should be as close to 0.7 as possible.**
    - **Print out a prompt message "[IN] Enter the number of Craps games to be played:" to accept the number of times the Craps game being played each time. Refer to the example input and output for more prompt messages and printout**
    - **Print out the number of times that the player wins**
    - **Print out the win probability each time.**
    - **Print out the probability that is most close to 0.7 among all the plays you made.**
  - **Repeat playing the given number of times and print out the number of times the player wins until the player would like to stop playing.**

Example of Input & Output

- **A line starting with [IN] is asking for input. Otherwise it is an output. The win probability should be different for each output.**

```
[IN] Enter the number of Craps games to be played: 100000
Number of Craps games won by the player = 69093;   Win probability = 0.69093
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69189;   Win probability = 0.69189
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69358;   Win probability = 0.69358
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69230;   Win probability = 0.6923
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69323;   Win probability = 0.69323
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69065;   Win probability = 0.69065
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69176;   Win probability = 0.69176
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 68938;   Win probability = 0.68938
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69304;   Win probability = 0.69304
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 68990;   Win probability = 0.6899
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69076;   Win probability = 0.69076
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 68931;   Win probability = 0.68931
[IN] Continue to play? (Y or y for yes): y
Number of Craps games won by the player = 69196;   Win probability = 0.69196
[IN] Continue to play? (Y or y for yes): n
The best win probability = 0.69358
```

Grading

- **The grade will be equal to**
  100-abs(1.0-best WIN probability/0.7)*1000.

Here, abs() is a function calculating the absolute value of a given number.
Hence, if the best WIN probability you get is 0.6993, your score will be 100-abs(1.0-0.6993/0.7)*1000
= 100-abs(1.0-0.999)*1000=99.

Grading Rules for TA

- **Have to check whether the original main() function is rewritten into a function named crapsFunc().**
- **Have to check whether the win probability of the player is close to 0.7**
- **Have to check whether the game can be replayed many times as shown in the output example.**
- **Must check the best win probability appears among the plays just made.**

Follow All Requirements

- Input formats
- Output formats
- All constraints on input data, especially not accepting invalid inputs
- Coding styles
  - Avoiding using variables which do not have expressive power. That is, a variable name should carry the meaning of the matter in which the variable intends to represent.

If you don't follow the requirements, up to 30% of the points for your lab will be deduced.

Rules for Program Submission

- **Put all the relevant files in the same folder.**
- **Name your folder SID_LabX, where ID is your student ID number and X is the number assigned to the lab. If a lab has N parts, N>1, then create N sub-folders with their names SID_LabX_N in the the folder SID_LabX.**
  - **For example, for Lab 2 with only one part and with student ID number 1041544, the name of the folder must be S1041544_Lab2. N is omitted if there is only one part.**
  - **Another example, similar to the above but Lab 2 has two parts. Then, you have to create a folder S1041544_Lab2 and two sub-folders S1041544_Lab2_1 and S1041544_Lab2_2**
- **Compress the folder into a file named SID_LabX.zip, for example, S1041533_Lab2.zip. Then, submit the compressed file**
- **If you violate this rule, your lab will not be graded. If graded other penalty will be applied.**