Computer Programming 1

Laboratory 07

Reference-to (recap)

Derived type reference-to (&)
It allows you to give multiple names to a variable (actually, each expression with an address)
If the original variable is changed, then its "alias" will also be changed;

- Obviously, if the alias is also changed then the original variable will also undergo changes

Reference-to (2) (recap)

```
int x = 5;
int & y = x;

x++;    // 6
cout << y << endl;   // 6
y++;    // 7
cout << x << endl;   // 7

int k = 20;
y = k;
cout << x << endl;   // 20
```

Pointer-to (recap)

Derived type: pointer-to (*)
A pointer contains the address of another previously created object;
A memory space (always the same) is reserved to a pointer (suitable for containing the address), but the space for the pointed object is not automatically reserved!

- To get the value of the pointer object, the operator dereference * must be used

Pointer-to (recap)

```
int x = 5;
int * y = &x;        // The operator & returns the memory address

x++;    // 6
cout << y << " " << *y << endl;   // 6
(*y)++;    // 7
cout << x << endl;   // 7

int k = 20;
y = &k;
cout << x << endl;   // 7
```

Pointer-to (recap)

```
int x = 5;
int * y = &x;        // The operator & returns the memory address

x++;    // 6
cout << y << " " << *y << endl;   // 6
(*y)++;    // 7
cout << x << endl;   // 7

int k = 20;
y = &k;
cout << x << endl;   // 7
```

Under the hood

The compiler assigns to each variable a memory are where the value of the variable is stored.

This area is identified by an address.

Source code

int y

Internal representation

A lookup table:
y is stored @ 0x103020

Memory:

Address	
0x103020	10

value of y

Under the hood

The compiler assigns to each variable a memory are where the value of the variable is stored.

This area is identified by an address.

Source code

int y

int& y (the address of y)
int * py (contains an address)
*py (the value at address)

Internal representation

A lookup table:
y is stored @ 0x103020

Memory:

Address	
0x103020	10

value of y

Under the hood

Source code

```
int y
int *p_y
```

Internal representation

A lookup table:
y is storead @ 0x103020
p_y is stored @ 0x103024

Memory:

value of y

Address	
0x103020	10
0x103024	0x103020

value of p_y

Function signature

- The signature of a function is compsed of (i) the name of the function, and (ii) the number, ordering and type of the formal parameters

- E.g.,

  Function: `int computeSum(int param1, int param2)`

  Signature: `computeSum(int, int)`

Note: (formally) <u>the return value of the function is not part of the signature even if, sometimes, it is considered to be part</u>

Variable's visibility (1)

"Each name that appears in a C++ program is only valid in some [...] portion of the source code called its **scope**."

https://en.cppreference.com/w/cpp/language/scope

Variable's visibility (2)

```
char letter = 'g';

void f() {
    char letter = 'f';
    cout << letter;
}

int main() {
    f();
    cout << letter;
    char letter = 'm';
    cout << letter;
}
```

Global variable

Local variable

Variable's visibility (2)

```
char letter = 'g';

void f() {
    char letter = 'f';
    cout << letter;          //f
}

int main() {
    f();
    cout << letter;  //g
    char letter = 'm';
    cout << letter;   //m
}
```

Global variable

Local variable

Variable's visibility (2)

```
char letter = 'g';

void f() {
    char letter = 'f';
    cout << letter;          //f
}

int main() {
    f();
    cout << letter;  //g
    char letter = 'm';
    cout << letter;   //m
}
```

Global variable

Local variable

Parameter passing

- **By value:** copy the value of the current parameter

changes are not reflected in the current parameter

- **By reference:** the parameter is a reference (&) to the current parameter

- **By pointer:** the parameter is the address of the current parameter

step by pointer value, but of course you can change the pointed variable

Function writing styles: parameters

- **Using global variables** (no parameter passing)

- **With side effects on arguments** (parameters passed by reference or pointers)

- **Purely functional** (parameters passed by value)

Parameter passing by Value

```
void f(int number) {
    number = 2;
    cout << number;    // 2
}

int main() {
    int number = 1;
    f(number);
    cout << number;    // 1
}
```

Parameter passing by Reference

```
void f(int & number) {
    number = 2;
    cout << number;    // 2
}

int main() {
    int number = 1;
    f(number);
    cout << number;    // 2
}
```

Parameter passing by Pointer

```
void f(int * pNumber) {
    *pNumber = 2;
    cout << *pNumber;        // 2
}

int main() {
     int number= 1;
    f(&number);
    cout << number;    // 2
}
```

Default arguments

- Used to provide optional parameters with default values

E.g.,

```
int max(int n1, int n2, int n3=0, int n4=0, int n5=0);
int main() {
    cout << max(1,2,3);                    // 3
}
int max(int n1, int n2, int n3, int n4, int n5) {
        ...
}
```

Exercise 1: Serie *pi*

- Write a program that calculates the result of the following series that approximates the value *pi* (π)
- The approximation value of this series N, is given by the user.

$$\sum_{i=1}^{N} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots + \frac{1}{N^2}$$

- Compute pi (π) by considering that $\sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$

Constraint: use at least two functions in your implementation

Exercise 2: Print uppercase (v2)

- Given a character in input, write a program that checks - with a function named *checkCharacter* - whether the character provided in input is a lowercase letter of the alphabet. If so, the program must print the uppercase character on the screen, using another function - named *convertCharacter* - for the conversion.
- Do not use any global variables
- Use passing by reference for the function *convertCharacter*

E.g., 'a' => 'A'

Types of implementations:
(1) Without using functions in the cctype library (e.g., isalnum, isalpha, islower, tolower, etc.)
(2) By using functions of the cctype library (e.g., isalnum, isalpha, islower, tolower, etc.)

Exercise 3: Liar's Dice (v2)

- Write a program that:
  - it rolls 10 6-sided dices by using the function *rand()* as presented below.
  - then the program asks the user to guess how many dices turned out to be 1.
  - If the number specified by the user corresponds to the number of 1 of the random dices, the user is the winner, and a message is printed.
  - Otherwise, the number inserted by the user is wrong, a message is printed and the program execution ends
- Do not use any global variables (instead use the passing by reference)

```
#include <cstdlib>
srand(time(NULL));          //seed value
int random_number = rand() % 6 + 1
```

For additional details:
http://www.cplusplus.com/reference/cstdlib/rand/

Exercise 4: Swap

- Given in input two real numbers (double), in two variables, write a program that uses a procedure to exchange the values of the two variables using the passage of parameters by pointers.

Exercise 5: Division

Given in input two integers, write a program that makes the mathematical operation “division” by using a function. The function returns the quotient as return value and the rest is returned by means of a reference parameter.

- In this program, you cannot use the symbols ‘/’ and ‘%’ for division (i.e., the division quotient and the rest needs to computed in a different way)

Exercise 6: seconds/minutes/hours

- Given as input three integers in three variables (seconds, minutes, hours), write a program that, with a procedure, converts a possible excess of seconds into minutes and a possible excess of minutes into hours.
- Use parameter passing by reference and/or pointer.
- (if possible) Use the function “division” implemented in the previous exercise

sec = 121;
min = 59;
hour = 2;
	=>	sec = 1;
		min = 1;
		hour = 3;

Exercise 7: Max (1)

- Given as input a set of 1 to 5 integers, write a function that returns the maximum value.
- Use default arguments for the second, third, fourth, and fifth inputs.

Constraint: use only fundamental data types

Exercise 7: Max (2)

- Given as input a set of 1 to 5 integers, write a function that returns the maximum value.
- Use default arguments for the second, third, fourth, and fifth inputs.

Variant 1: ask the user the numbers as input
Variant 2: check the number inserted by the user, with the aim of avoiding integer overflow
Variant 3: write a program that iteratively requires to the user if stop the execution of if find the max in another set of numbers
Variant 4: use arrays

Exercise 8: Sort (1)

- Given as input three positive integers, in three variables (n1, n2, n3), write a program that, with a procedure, "re-orders" the numbers in ascending order by using the passage of parameters by reference.

```
n1 = 3;
n2 = 7;      =>      n1 = 2;
n3 = 2;              n2 = 3;
                     n3 = 7;
```

Exercise 8: Sort (2)

- Given as input three positive integers, in three variables (n1, n2, n3), write a program that, with a procedure, "re-orders" the numbers in ascending order by using the passage of parameters by reference.

n1 = 3;		n1 = 2;
n2 = 7;	=>	n2 = 3;
n3 = 2;		n3 = 7;

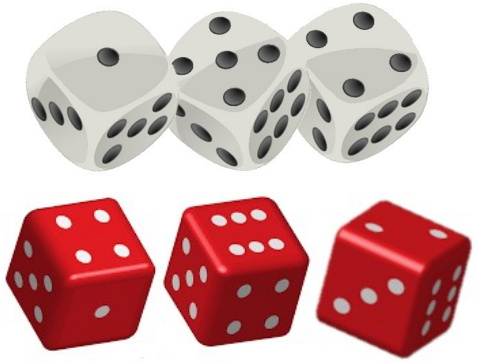
Variant 1: extend the previous implementation to work with more than 3 integers, by defining a max integer N
Variant 2: ask the number of integer to the user
Constraint: use arrays

Exercise 9: RisiKo! 3vs3

- Write a program that simulates a 3-on-3 attack on RisiKo!. Roll 3 dices to 6 faces for the attacker and 3 dices for the defender. Compare the highest dice of the attacker against the highest of the defender, the middle of the attacker against the middle of the defender and the lowest of the attacker against the lowest of the defender. Finally, declare the number of clashes won by the attacker and the defender.
- It is requested to develop (at least) a version of the program that:
  - It uses a function for roll the dice – random number –, and three procedures respectively for sorting, swapping and comparing the rolled dices
  - It uses the passage of parameters by reference at least for 2 procedures

Defender

Stricker



5 < **6** => Stricker

4 = **4** => Defender

1 < **2** => Stricker

Exercise 10:Swap v2

- Given in input two integers (short), write a program that, with a procedure, exchanges the 8 bits less significant in the two numbers.
- It is requested to develop (at least) a version of the program that uses the passage of parameters by pointers for the procedure.

```
1855(0000011100111111)      1816(00000111<u>00011000</u>)
1048(0000010000011000)  =>  1087(00000100<u>00111111</u>)
```