CityU
香港城市大學
City University of Hong Kong
專業 創新 胸懷全球
Professional·Creative
For The World

CS2313 Computer Programming

LT12 – Final Revision

1-Basic Concept

```
#include <iostream> //include the file
using namespace std;
int main() //main function (return integer)
{
   int a; //function body
   cout << "Hello, world!\n";
   return 0;
}
```

- Tokens
  - *Keywords, identifiers*
  - *string constants, numeric constants*
  - *punctuators*

- Variable and constant
  - Name, type, scope, address
  - Should be initialized before using
  - Variable name/identifier should not be identical with the keyword
  - Data type: **`int/float/double/char/bool`**…
  - Constant
    - The constant value could not be changed during program execution
    - Must be initialized in the declaration

```
{
……
int i;
for(i=0;i<10;i++)
{
……
}
cout<<i; ……
}
```

```
for(int i=0;i<10;i++)
{
……
}
cout<<i; //i could not be printed
```

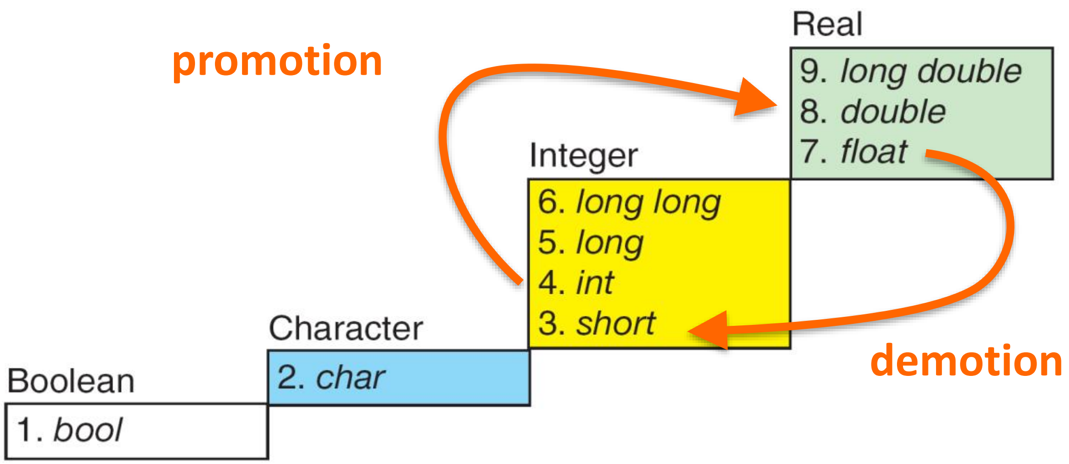
2-Console I/O(Input/Output)

- cin and cout
  - cout : Insertion Operator <<
  - cin : Extraction Operator >>

- cout – change the format of the output
  - cout.width()
  - setw()
  - setprecision()
  - fixed
  - scientific
  - #include <iomanip>

3-Type conversion

- Promotion and demotion



- Type conversion
  - **Implicit** type conversion
  - **Explicit** type conversion

- Implicit type conversion
  - binary expressions (e.g. `x + y`): lower-ranked operand is promoted to higher-ranked operand.
  - assignment (e.g. `x = y`): right operand is promoted/demoted to match the variable type on the left.

- Explicit type conversion
  - `int i = (int)3.9;`
  - `double j = (double)i;`

4-Operators

- Two important operators
  - /: be careful about the data type
  - %: how to use the remainder operator
- Precedence and Associativity of Operators
- Assignment operator = and equality operator ==
- Generic form of efficient assignment operators

variable op= expression;

where *op* is operator; the meaning is

variable = variable op (expression);

4-Operators (a++ and ++a)

Two things happen:

(1) Compute the value of the expression `a++` or `++a`.

(2) Increment `a` by `1`.

a++

do (1) before (2).

Therefore, the value of a++ is equal to old value of a.

++a

do (2) before (1).

Therefore, the value of ++a is equal to the incremented value.

5-Comparative Operators

- Operators
  - `==, !, !=, >, >=, <, <=, &&, ||`
  - `true/false`
  - `&&` or `||`?

x	y	x&&y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x||y
true	true	true
true	false	true
false	true	true
false	false	false

x	!x
true	false
false	true

6-Conditional Statement

- `if()/else if()/else`
  - Two way selection
  - Compound statement with braces {}
  - Nested `if` statement
  - The else if or else part is executed only if all the preceding conditions are false
  - Be careful about the equality operator
  - Semicolon
    - Belong to the expression statement, not to the if...else statement

switch, case, break

- Evaluating the `switch` expression returns an integer type (`int`, `long`, `short`, `char`).

- Go to the `case` label having the constant value that matches the value of the `switch` expression; if a match is not found, go to the `default` label; if `default` label does not exist, terminate the `switch`.

- Terminate the switch when a `break` statement is encountered.

- If there is no `break` statement, execution "*falls through*" to the next statement in the succeeding case.

7-for loop

- expr1, expr2, expr3

```
for (expr1 ; expr2; expr3) {
        statements;
}
```

expr2: Becomes false, the loop ends.
expr1: Executed before entering the loop. Often used for variable **initialization**.
expr3: For each iteration, **expr3** is executed after executing loop body. Often used to update the counter variables.

expr1:	**i=0**	**i=1**	**char* str=ptr**
expr2:	**i<arraysize**	**i<=arraysize**	***str!=‘\0’**
expr3:	**i++**	**i++**	**str++**
	array	**array**	**cstring**

- Nested `for` loop
  - Outer loop
  - Inner loop
  - Within the for loop, we can have if/else statement
  - Compound statement {}

```
for ( int i=0; i<K;i++)
    for (int j=0; j< L; j++)
……
```

for each *i*, loop over *j* (from 0 to L-1)

Be careful about the extra semi-colons after for loop

8-while loop

- General form of **`while`** statement (*repetition* statement)

```
while (logical expression) {
    statements
  }
```

  - `logical expression` is evaluated first.
  - If `true`, `statements` in the loop body are executed.
  - Then `logical expression` is evaluated again and if still evaluated to `true`, the loop repeats; otherwise, the loop terminates.

9-do Statement

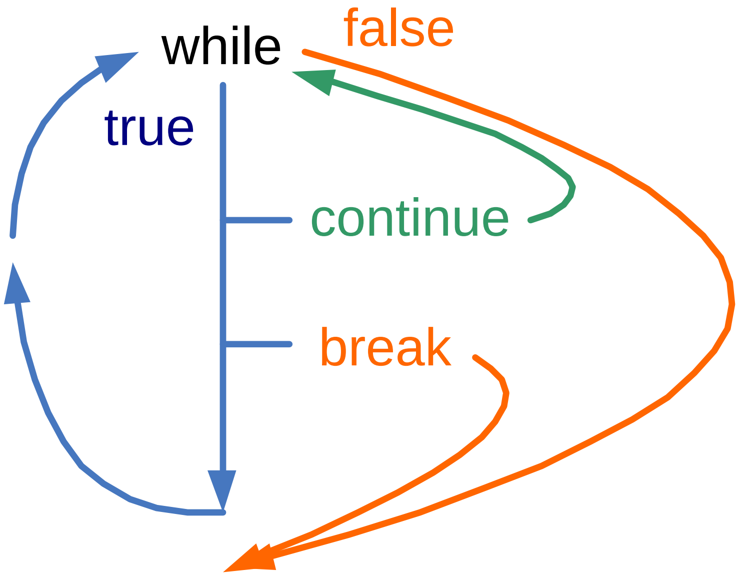
- General form of **`do`** statement (*repetition* statement)

```
do {
   statements
} while (expression);
```

- Semantics:
  - `statements` are executed first; thus the loop body is executed at least **once**.
  - Then `logical expression` is evaluated and if `true`, the loop repeats; otherwise, the loop terminates.

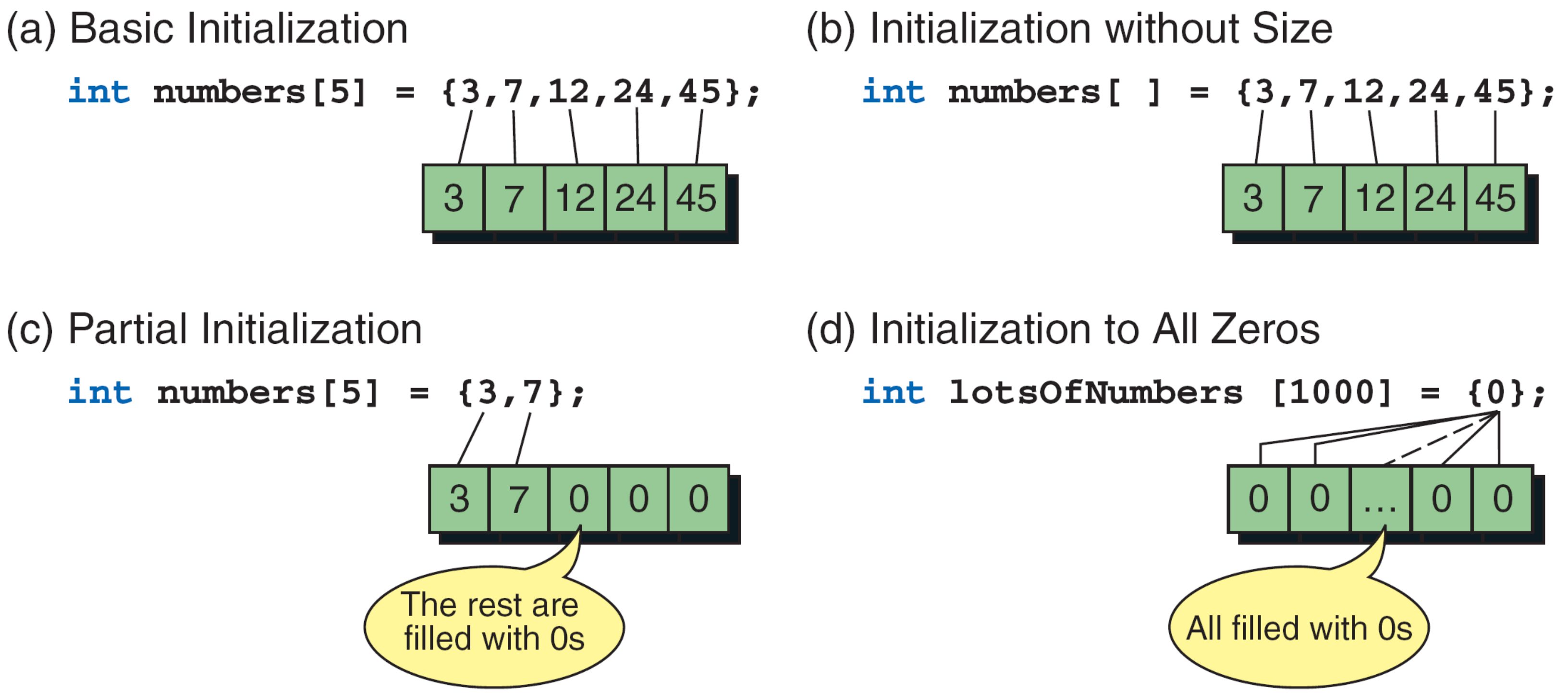
10-break/continue

- the **`break`** statement causes an **exit** from the innermost enclosing loop or `switch` statement
- `continue` statement causes the current iteration of a loop to **stop** and the next iteration to begin immediately.



11-Array

Declaration and initialization of the array



(a) Basic Initialization

```
int numbers[5] = {3,7,12,24,45};
```

(b) Initialization without Size

```
int numbers[ ] = {3,7,12,24,45};
```

(c) Partial Initialization

```
int numbers[5] = {3,7};
```

(d) Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```

Accessing array elements

- Access the value of the i-th element
  - **`Array[i-1]`**
- Range of the index
  - from 0 to **arraysize-1**
- Compare two arrays
- Visit the array using loop

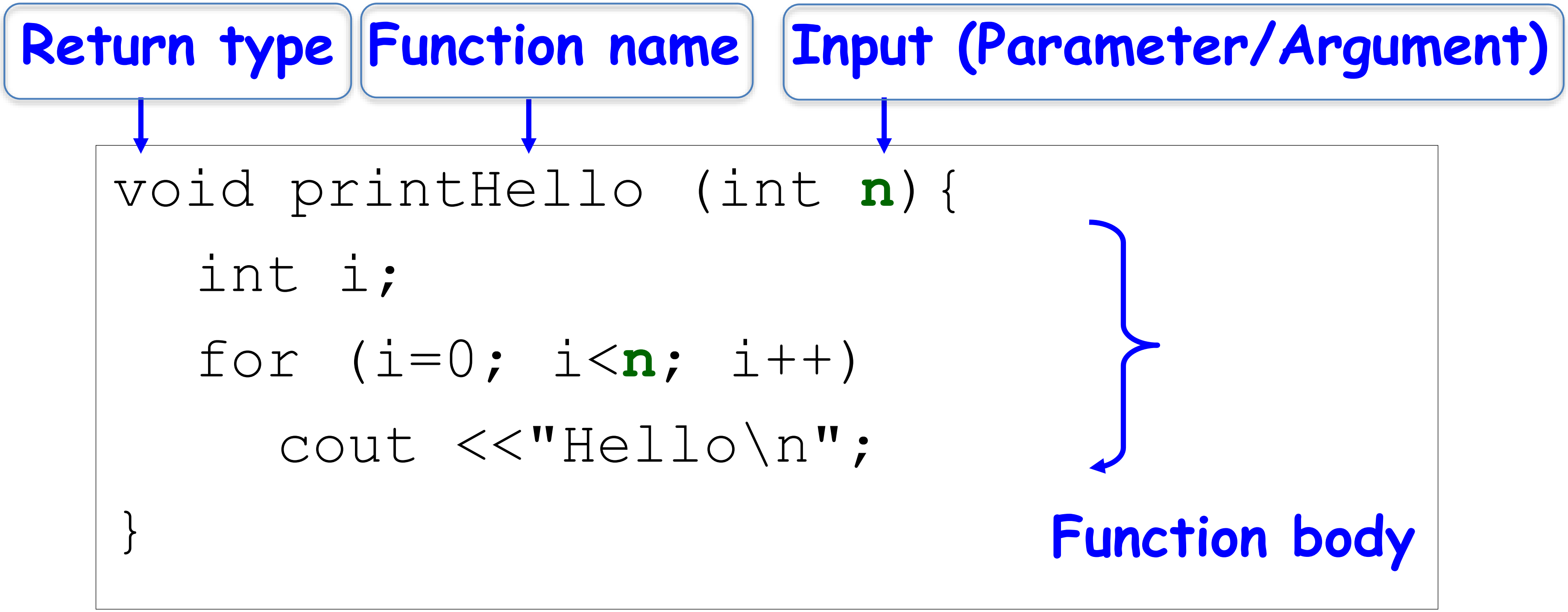
```
int sum = 0;
//assuming we have the array with 10 elements
(mark[10])
for (i=0; i<10; i++) {
    cout << mark[i];
    sum = sum + mark[i];
}
```

12-Bubble Sort

```
#include <iostream>
using namespace std;
#define n 10
void main() {
  int a[n], j, k, tmp;

  cout <<"Input" << n <<" numbers: ";
  for (j=0; j<n; j++)
    cin >> a[j];

  for(j=0; j<n-1; j++)   // outer loop
    for(k=n-1; k>j; k--) // bubbling
      if (a[k]<a[k-1]) {
        tmp    = a[k];   // swap neighbors
        a[k]   = a[k-1];
        a[k-1] = tmp;
      }

  cout << "Sorted: ";
  for(j=0; j<n; j++)
    cout << a[j];
}
```

How to swap the values using *call by reference* function

13-function



Return type

Function name

Input (Parameter/Argument)

```
void printHello (int n){
   int i;
   for (i=0; i<n; i++)
      cout <<"Hello\n";
}
```

Function body

Return type: void (no return), int, float, char*……

The function is invoked and the program control is passed to that function; when the function ends, program control is returned to the statement immediately after the function call in the original function.

- Call a function
  - If no value is returned
    - `printHello (3);`
  - If the function returns a value
    - `cout<<function(input)`
    - `int i=function(input)`
  - Syntax error
    - `int printHello (3);`
    - `printHello (int x);`

14-Function Prototype

- Function Prototype
  - Specifies the function name, input and output type.
    - `int findMax (int, int);`
    - `int findMax (int a, int b);`
    - `void f(void);`
  - The function can be defined later.
  - The prototype should exactly match the function definition (const).
  - Define the function prototype in class.

15-Call by Value

- In call by value, value could only be updated inside the function
- Update the value outside of the function- return the value

```
void f (int y){
  y=4;
//modify y in f(), not the one in main()
}
void main(){
  int y=3;
  f(y);
  cout << y; //print 3, y remains unchanged
}
```

In `f()`, `y` in `f()` is modified.

However, `y` in `main()` is not affected.

16-Call by Reference

- In call-by-value, only a single value can be returned using a **return statement**.
- There are two ways for call by reference
  - More than one variables can be **updated**, achieving the effect of returning multiple values.

Examples of call by reference

```
void swap(int &a, int &b)
{
     int temp;
     temp=a;
     a=b;
     b=temp;
}
int main()
{
     int x=3,y=5;
     swap(x,y);
}
```

```
void swap(int *a, int *b)
{
     int temp;
     temp=*a;
     *a=*b;
     *b=temp;
}
int main()
{
     int x=3,y=5;
     swap(&x, &y);
}
```

17-Recursive Function

- Recursive Function
  - A recursive function is a function that calls itself.
  - A base case should eventually be reached, at which time the breaking down (recursion) will stop.

- How to define Recursive Function
  - Find the relationship between the problem and the sub-problem defined in the same way.
  - Break down the problem into smaller or simpler sub-problems of the same type.
  - Find the termination condition (base case)

18-class and object (concept)

- With class, variables and their directly related functions can be grouped together to form a new data type
  - definition of program component
  - consists of member variables and member functions
  - Member variable : variable belong to class
  - Member function: function primary designed to access/manipulate the member variable of the class

- Object is an instance of class, i.e. *class* is a blue-print and its product is its *object*. Objects can also form an array.
- Private/public: Private members can be accessed only by member functions (and friend functions) of that class, not from outside

19-function and variables in class

- In C++, a class definition commonly contains only the *prototypes* of its member functions (except for inline functions)
- Use *classname::functionName* to define the member function (method) of particular class.
- Member Function can directly get access to the variables and is called using the dot operator
- By default, all members of a class are private

20-constructors

- A *constructor* is a member function that is automatically called when an object of that class is declared
- Special rules
  - A constructor must have the same name as the class
  - A constructor definition cannot return a value
- A constructor cannot be called in the same way as an ordinary member function is called.
- In constructor, the variables are not required to be declared again.
- More than one versions of constructors are usually defined (overloaded) so that objects can be initialized in more than one way.
- Default constructor (value=0; pointer=NULL;)

21-friend and const

- A friend function of a class is *not* a member function of the class but has access to the private members of that class
- In function prototype, indicate the *friend* function
- In call-by-reference, if the function is not supposed to change the value of the parameter, you can mark it with a const modifier
- The prototype should be consistent with the function implementation.

22-overloading operators

- Motivation
  - `Circle C1, C2;`
  - `cout<<C1;`
  - `Circle C3=C1+C2;`

```
friend Circle operator+(const Circle &c1,const Circle &c2);
Circle operator+(const Circle &c1,const Circle &c2){
    Circle c3(c1.radius+c2.radius);
    return c3;
}

friend ostream &operator << (ostream &outs, const Circle &c);
ostream &operator << (ostream &outs, const Circle &c){
        outs << c.getArea();
        return outs;
```

23-char

- Character data type
  - Single quotes
  - A **`char`** type is represented by an **integer**
  - ASCII Table to convert char to integer
- **`Char`**
  - Comparison
    - `char c1;if(c1==‘a’)…;`
  - Convert to lowercase/uppercase
    - `c3=c1+'a'-'A';`
    - `c4=c2+'A'-'a';`
  - Check if it is a lowercase or uppercase
    - `(c1>='a'&&c1<='z')||(c1>='A'&&c1<='Z')`

24-cstring

🕮cstring

- `'\0'`: end-of-string
- A character of array of size **`n`** can store a string with maximum length of **`n-1`**.
- When visiting a cstring, do not access all the elements. Just visit the element until reaching the null character `'\0'`
- At the end of the cstring, add `'\0'` (for example, in cstring manipulation)
- `char str[20]="Hello World";` does not mean the string size is 20.

🕮Input/output

- `cin.getline(stringname,size);`
- `cout<<stringname;`

Common `cstring` Functions in `<cstring>`

Function	Description	Remarks
`strcpy(dest,src)`	Copy the content of string src to the string dest.	No error check on the size of dest is enough for holding src.
`strcat  (dest,src)`	Append the content of string src onto the end of string dest.	No error check on the size of dest is enough for holding src.
`strcmp(s1,s2)`	Lexicographically compare two strings, s1 and s2, character by character.	0: s1 and s2 is identical >0: s1 is greater than s2 <0: s1 is less than s1.
`strlen(string)`	Returns the number of characters (exclude the null character) contain in the string.	

25-pointer

- Basic concept
  - A pointer is a variable which is designed to store the address of another variable.
  - Its type is determined by the type of variable it points to.
- * operator
  - Defines a pointer
  - Dereference operator
- & operator
  - Address operator (get the address)
  - Reference type

26-pointer and array

- An array variable without a bracket and a subscript represents the starting address of the array.
- An array variable is essentially a constant pointer.
- Pointer **arithmetic**

```
int a[10]={0};
int *p=a;
p++;p--;*p=5;cout<<*(a+1);…
```

`a+i` represents the address of `a[i]`

27-pointer and cstring

- How to access each element in the string?

```
int count(char *s, char c)
{
  int occurrence=0;
  for (char * pi=s; *pi!='\0'; pi++)
  {
      if (*pi==c)
          occurrence++;
  }
  return occurrence;
}
```

```
int count(char s[100], char c)
{
   int occurrence=0;
   int i; //counter
   for (i=0; s[i]!='\0'; i++)
   {
        if (s[i]==c)
            occurrence++;
   }
   return occurrence;
}
```

28-Pass array/string to function

- 🕮 Passing Array to the function
  - – Array size can be specified

```
f(list,size)

void f(int list[], int size)
{
//function body
}
void f(int *list,  int size)
{
//function body
}
```

- 🕮 **Passing String to the function**
  - – No string size as input

```
f(str)

void f(char list[])
{
//function body
}
void f(char *list)
{
//function body
}
```

29-new and delete

Variable-define array size

```
int* result = new int[arraysize];
// Allocate
delete [] result;
// Deallocate after using the memory

int* p = new int; // Allocate
delete p; // Deallocate
```

30-File I/O(Input/Output)

- File stream class
  - #include <fstream>
  - `ifstream: stream class for file input`
  - `ofstream: stream class for file output`
- Objects
  - `ifstream fin;`
  - `ofstream fout;`
- File open and close
  - `fin.open(“xxx.dat”);fout.open(“xxx.dat”);`
  - `fin.close(); fout.close();`
- Read and write
  - `fin >> x;`
  - `fout << x;`

Common Mistakes

- Cannot differentiate a++ and ++a
- Cannot differentiate << and >>
- Cannot differentiate == and =
- Cannot differentiate && and &
- Cannot differentiate keywords and identifiers
- Cannot differentiate cstring and traditional array (int, float)

Common Mistakes

- Scope: use the variable that is outside of the scope
- When the array size is a variable, forget the dynamic memory allocation
- Directly compare two array with ==, instead of comparing each element
- Revise the value of constant in the program
- Forget about the break in switch

Common Mistakes

- Ignore the type conversion
- Count the array from 1 instead of 0
- Forget the end of cstring: ‘\0’
- Use call-by-value instead of call-by-reference when attempting to change the values of the variables

Useful Suggestions

- Pay attention to the details
- if…, else if…, else…
- for loop
- while, do while loop
- Pointer and array
- Pointer **arithmetic**
- Recursive functions
- Call by reference

Useful Suggestions

- Read and write files
- Cstring: ‘\0’
- Dynamic memory allocation
- Bubble sort
- Class
  - Member functions, constructors
  - Overloading operators
  - Friend functions

Useful Suggestions

- Review the slides and programs (lab session)
- Read the requirements of the questions carefully
  - In some questions, using cstring *function* is not allowed
- Read the hints carefully
- Some comments are provided in the program to help you solve the problem
- In some questions, the skeleton code has been already given to help you solve the problem