Conventional Legend:

[*C*] used in C language

[*F*] function

[#include<>] those keywords or instructions need head files

ORANGE represents personal and uncertain deduction

RED represents those rules that may be changed in new standards

GREEN contents of notes and comments

BLUE represents keywords, other words with capital initials are specifically edited contents

const DataType DataName

# Abbreviation and Definition:

|  |  |
| --- | --- |
| API | Application Programming Interface |
| C++/CLI | One of the two fundamental kinds of C++ applications, short for Common Language Infrastructure, target the .NET Framework and Windows Forms application |
| CLR | Common Language Runtime (see p2) |
| ECMA | European Computer Manufacturers Association |
| IDE | Integrated development environment |
| ISO/IEC | One of the two fundamental kinds of C++ applications, short for International Standards Organization/ International Electrotechnical Commision |
| MFC | Microsoft Foundation Classes |
| OOP | Object-oriented programming |
| PPL | Parallel Patterns Library; make it easy to program for using multiple processors |
| STL | Standard Template Library |
| MSIL | Microsoft Intermediate Language |
| JIT | Just-in-time |
| CTS | Common Type System |

# Signs, Symbols and Operators:

|  |  |
| --- | --- |
| //… | Used to add comments, not be executed by program |
| /\*…\*/ | Used to add comments, not be executed by program |
| :: | Twn colons; scope resolution operator |
| {} | Curly brace and closing curly brace; for enclosing a block of directives; one can define private variables inside the braces |
| ; | Semicolon; marks the end of a statement |
| \ | used at the end of a character sequence and end of a line to indicate continuing characters in the next line 续行符 |
| # | preprocessor sign 预处理器操作符，其后的指令和变量全部对应预处理器的一些操作，如“宏” |
| new | new T(X) OR new T{X}; new T[X]  allocate space for a dynamic datum; T as data type, X as initial value and can be omitted;  ClASS: new T(); 如果使用默认构造函数不能省略括号  ARRAY: T as data type of an array, X as dimensions, one cannot assign initial values in this case; the expression returns an address of the space or “NULL” if the process fails (so usually this expression is used to assign an address to a pointer variable). |
| delete | delete P OR delete [ ] P  delete the space allocated by “new” operation; P as the pointer pointing to the space; if the space is for arrays, one should add “[ ]” |

# Data Types

|  |  |  |  |
| --- | --- | --- | --- |
| TYPE | SIZE IN  BYTES | RANGE OF VALUES | EXAMPLES |
| bool | 1 | true or false | true, false |
| [signed] char | 1 | -128 to 127 | ‘A’, ‘Z’, ‘8’, ‘\*’ |
| signed char | 1 | -128 to 127 | ‘A’, ‘Z’, ‘8’, ‘\*’ |
| unsigned char | 1 | 0 to 255 | ‘A’, ‘Z’, ‘8’, ‘\*’ |
| wchar\_t | 2 | 0 to 65,535 | L‘A’ , L‘Z’, L‘8’, L‘\*’ |
| short [int] | 2 | -32,768 to 32,767 |  |
| unsigned short [int] | 2 | 0 to 65,535 |  |
| [signed] int | 4 | -2,147,483,648 to 2,147,483,647 | - 77, 65, 12345, 0x9FE, 020[=(20)8], 0X20[=(20)16] |
| unsigned [int] | 4 | 0 to 4,294,967,295 | 10U, 64000u |
| long [int] | 4 | -2,147,483,648 to 2,147,483,647 | - 77L, 65L, 12345l |
| unsigned long [int] | 4 | 0 to 4,294,967,295 | 5UL, 999999UL, 25ul, 35Ul |
| long long | 8 | -9223372036854775808 to  9223372036854775807 | - 777LL, 66LL, 1234567ll |
| unsigned long long | 4 | 0 to 18446744073709551615 | 55ULL, 999999999ULL, 885ull, 445Ull |
| float | 4 | ±3.4 × 10± 38 with  approximately 7 digits accuracy | 3.14f, 34.506F, 3.14e-4F |
| double | 8 | ±1.7 × 10± 308 with  approximately 15 digits accuracy | 1.414, 2.71828, 3.14e-4 |
| long double | 8 | ±1.7 × 10± 308 with  approximately 15 digits accuracy | 1.414L, 2.71828l, 3.14e-4L |

**Special Types:**

|  |  |
| --- | --- |
| string | #include<string> |
|  |  |

# Operators and Sequence of Calculation

|  |  |  |
| --- | --- | --- |
| :: |  |  |
| (); []; ->; . |  |  |
| ! (NOT); ~(bitwise operator, NOT);  + (unary) - (unary) ++; - - & (unary) \* (unary) (typecast)  static\_cast const\_cast dynamic\_cast reinterpret\_cast  sizeof new delete typeid decltype |  |  |
| .\* (unary) -> \* |  |  |
| \*; /; % |  |  |
| +; - |  |  |
| <<(bitwise operator); >> (bitwise operator) |  |  |
| <; <=; >; >= |  |  |
| ==; != (equal; not equal) |  |  |
| & (bitwise operator, AND) |  |  |
| ^ (bitwise operator, XOR) |  |  |
| | (bitwise operator, OR) |  |  |
| && (AND) |  |  |
| || (OR) |  |  |
| ?: (conditional operator) | X? Y:Z | if X is true, expression=Y; if X is false, expression=Z |
| =; \*=; /=; %=; +=; -=; &=; ^=; |=; <<=; >>= |  |  |
| , |  |  |

# Ranking of Data Types for Conversion

|  |
| --- |
| long double |
| double |
| float |
| unsigned long long |
| long long |
| unsigned long |
| long |
| unsigned int |
| int |

# Escape Sequence

|  |  |  |
| --- | --- | --- |
| ESCAPE SEQUENCE | WHAT IT DOES | ASCII |
| \a | Sounds a beep | 007 |
| \n | Newline | 010 |
| \t | Tab | 009 |
| \b | Backspace | 008 |
| \r | return to the start point of the current line | 013 |
| \f | move to the start point of next page | 012 |
| \v | jump vertically | 011 |
| \\ | Backslash, = “\” | 092 |
| \’ | Single quote | 039 |
| \” | Double quote | 034 |
| \0 or \000 | empty character(NULL) | 000 |
| \? | Question mark | 063 |
| \ddd | output character with the ASCII of (ddd)8 |  |
| \xhh | output character with the ASCII of (hh)16 |  |

# Directive/Declaration References:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Preprocessor Directives** ( no “;” at the end of line) | | | | | |
| #include<X> | Insert the contents of fileX into the program source file before compilation (Examples of X: cmath, iostream, iomanip, string ) | | | | |
| #include“X” | <>for standard header files, “”for custom header files | | | | |
| #define X Y | [*C*] define a public character constant, X as name, Y as constant value or character sequence；中文名称为宏，实际上代码中的任何X字符串都会被Y替代，无论是左值还是右值，因此X一般全部取大写字母命名以区分 | | | | |
| #define X(Y) Z | [*C*] define a simple function, X as function prototype, Y as parameters, Z as expression of the function；同样，宏代码只是被替换，因此X不完全遵照C++函数定义的语法，或干脆不是一个函数。1）X中的参数是无需定义类型的任意类型，即只需要一个名称用于在Z中替换即可。2）Z实际上是一串用于替换的字符，所以无需（不是不能）像函数体一样加{}。但是Z的内容仍必须符合程序语法，且须注意替换后不要产生歧义。 | | | | |
| #X | 预处理符号的单独使用：将X的名称或表达式转换为字符串，形如“X”。注意不是将X变量的内容转换为字符串，因为预处理器不执行语义，只是做文字表面的替换工作。（这个用法是否可以被“X”完全替代，或这种写法是否是为了用于避免“X”产生的歧义——到底只是表示一个“X”字符串还是需要将X宏替换，尚不明确） | | | | |
| #ifdef (or #ifndef)  #else  #endif | conditional compilation, see p131 | | | | |
| #if …  #else  #endif | conditional compilation, see p131 | | | | |
| #pragma once | unofficial include guard to make sure this source file can only be compiled once; usually using #ifndef block is better for compatible reasons | | | | |
| **Statements and Data** | | | | | |
| using std::X; | | | Use X as the X from std namespace without specifying namespace anymore | | |
| using namespace X; | | | use the keywords of namespace X without specifying namespace anymore (Examples of X: std, ) | | |
| using Y = X; | | | similar to “typedef X Y”, define custom type Y as equivalent to the X type; X is standard type or class/structure | | |
| T X1=Y1, X2=Y2,…; | | | define a variable, T as type, X as name, Y as initial value or an expression | | |
| T X1(Y1), X2(Y2),…; | | |
| typedef X Y; | | | define the type Y as equivalent to the X type; X is standard type, like int/double; Y is the custom name. (see p225)  使用方法：（例子为指向函数的指针类型）  ①先按照定义变量或函数的方法写出定义语句(int (\*p)())  ②将变量名换成新类型名（习惯全大写以区分）(int (\*PF)())  ③在最前面加typedef (typedef int (\*PF)())  ④然后可以重用新类型名去定义真正的变量(PF p1,p2) | | |
| union T  { X1 Y1;  X2 Y2;…} Z1, Z2 | | | define a union type “T”; X and Y are the members’ types and names. Z as the name of the union variable or union array (Z or T can be omitted) | | |
| enum X{Y1=y1, Y2=y2, …} Z1, Z2, …; | | | Define an enumeration type; Z as variable name, Y as possible options of Z (Y can be characters), y as represented values by Y (y are numbers, y can be omitted to use default values), X as the type name; Z can only be assigned by Y1, Y2 not y1, y2, but Y is processed as y. | | |
| const X Y1=Z1…; | | | define a constant variable, similar to variable | | |
| mutable X Y1=Z1… | | | define a mutable variable, i.e. a normal/general variable, thus “mutable” can be omitted by default | | |
| template<typename T1, typename T2, …> … | | | define a template; T1, T2 as the names of the custom data types (or called “type parameter”); one can use T1, T2 to define a function like standard data types (int, float, double…). (“typename” is a key word and can be equally replaced by “class”; note that this sentence must be followed by function declaration and with a “;” as a whole directive) [see details on p102，函数模板使用时，因为提供了参数，所以编译器可以自行推断出类型，所以函数的调用可以省略实例化部分，即省略F<X>(…)中尖括号部分] | | |
| auto | | | used in front of the type name in the declaration of data (default, can be omitted)  (from c++11, the definition of auto has been changed) | | |
| static | | | used in front of the type name in the declaration of data (If not given an initial value, it will be set 0 or ‘\0’ (not “space”) automatically. p116); (the static global variable cannot be extended to other files; the static local variables are used in functions to save data from last call); ( the static local variable in a function is given an initial value (by user or automatically) only once at the first call; the auto/default variable is assigned at every call) | | |
| register | | | declare variables stored in register to accelerate the calculation (not needed for modern compiler) | | |
| extern | | | used in front of the type name in the declaration of data to extend the availability of data to all the source files | | |
| (X)Y | | | [*C*][*F*] convert Y to type X; X as type (can also be pointer type, like “char\*”), Y as variable name or expression; the conversion renders intermediate quantity and does not change the value of Y. (all the types including user defined types, like enumeration can serve as X) | | |
| X(Y) | | | [*C++*][*F*] convert Y to type X; X as type (can also be pointer type, like “char\*”), Y as variable name or expression; the conversion renders intermediate quantity and does not change the value of Y | | |
| **Decision Structure** | | | | | |
| if | | | if (…) …; | | |
| if (…) …;  else …; | | |
| if (…) …;  else if (…) …;  else if (…) …;  else …; | | |
| switch | | | switch(X)  {case Y1: …; break;  case Y2: …; break;  …;  default: …; break;  } | switch(X)  {case Y1:  case Y2: …; break;  …;  default: …; break;  } | |
| **Loop** | | | | | |
| while  (decide first execute later) | | | while (…)  {…  } | | while (…) … |
| do-while  (execute first decide later) | | | do  {…  } while (…); | | do …  while (…); |
| for | | | for (X; Y; Z) …;  (X sets initial value of loop variable; Y as loop condition; Z increases or decreases the loop variable; X/Y/Z can be omitted and defined elsewhere) | | |
| break | | | jump out of the loop to execute following instructions | | |
| continue | | | end the current loop and return to the start of the loop | | |
| **Function** | | | | | |
| T X(Y1 Z1, Y2 Z2,…)  {…  return (…)  } | | | T as the type of function; X as the name of function; Z as the parameters of the function or can be “void”; Y as the types of parameters. If T=void, return (…) can be omitted. (one can define the default value of Z in definition or declaration, the defined Z should be placed at the end of the parameters list. p103) | | |
| inline T X(Y Z,…) | | | define a inline function; inline function should be a simple and frequently used function (with no loop inside). | | |
| static | | | used in front of the type name in the definition of function, this function cannot be used by other files | | |
| extern | | | used in front of the type name in the definition of function (default, can be omitted); other files can use this function with declaration of function preceded by “extern” (can also be omitted). | | |
| **Array** | | | | | |
| T X[Y]={Z1, Z2…..} | | | define an array, T as data type, X as the name of array, Y as the dimensions of array, Z is the initial value of array (those elements without initial values will be set 0 automatically) | | |
| T X[Y1][Y2]={{Z1, Z2…},{…}…} | | |
| strcat(X, Y) | | | [#include<string>][*F*][*C*] combine Y with X, X and Y are character arrays | | |
| strcpy(X, Y, Z) | | | [#include<string>][*F*][*C*] copy Y into X, X is a character array, Y can be either a character array or a string, Z is the number of characters to be copied from Y plus a ‘\0’. (if Z is omitted, then Y is wholly copied into X) | | |
| strcmp(X, Y) | | | [#include<string>][*F*][*C*] compare the X and Y; X and Y can be either character arrays or strings. p154 | | |
| strlen(X) | | | [#include<string>][*F*][*C*] output the length of string stored in array X, excluding ‘\0’. | | |
| **Pointer** | | | | | |
| T \*X1, \*X2,… | | | define pointer variables which can point to variables; T as the data type of the pointed variable; X as the name of the pointer variable. | | |
| T (\*X)[Y] | | | define a pointer variable which can point to an 1D array; T as the data type; X as the name of pointer variable; Y as the dimension.  理解方式：\*X是X的解引用，表示的是一个一维数组的名称，一维数组名称也是指向该数组的指针，因此X是指向指针的指针，即指向一维数组的指针 | | |
| T \*X[Y] | | | define a pointer array whose elements are all pointer variables. T as the type of the variables pointed to by the elements of the array. | | |
| T \*\*X (or T \*(\*X)) | | | define a pointer variable pointing to a T type pointer. | | |
| char \*X= “Y” | | | define a character pointer variable, X as the name of pointer variable, Y as the contents of string; X points to the address of the first element of Y; the string can also be output by “ cout<<X” as a whole, like using X as a string variable. (the value of a string constant is the address of its first element) | | |
| T (\*X)(Y1 Z1, Y2 Z2, …) | | | define a pointer variable pointing to a function; T as the type of function; X as the name of pointer variable; Y as the types of the virtual parameters of the function; Z as the virtual parameters and can be omitted. Use function name to assign address to X and X can replace the name of the function.  理解方式：函数指针指向函数的入口地址，函数名即该函数的入口地址变量；赋值给指针后，指针变量可以代替函数名，道理类似于数组。此处对函数指针介绍不完整，详参中高阶知识积累。 | | |
| T \*X(Y1 Z1, Y2 Z2…) | | | define a function which returns a pointer constant pointing to a “T” type variable. | | |
| const T \*X | | | define a pointer variable pointing to a “T” type constant (constant variable must be pointed to by this kind of pointer variable; however, this kind of pointer variable can be used to point to a general variable, but one cannot use the pointer variable to change the value of the variable ) | | |
| T \* const X | | | define a constant pointer variable pointing to a “T” type datum, one cannot change the address stored in this pointer variable after it was assigned | | |
| **Reference** | | | | | |
| T &X=Y | | | define a reference for Y; T as the data type; Y as the name of an existing variable; X as the alternative name of Y. X and Y are the same variable with two names. | | |
| T X(Y1 &Z1, Y2 &Z2…) {} | | | define a function using reference as virtual parameters. Using this method can change the real parameters of the function directly. | | |
| const T &X | | | define a constant reference; (usually used as virtual parameters of functions, one cannot change the value of X in the function) | | |
| T& X(…) {} | | | define a function which returns “T” type reference. (usually, the returned value of a function is a temporary constant; using this method can return global variables or other global objects, which means the results of the function can be assigned or used like a variable; in this case, the user usually does not care about the returned value of the function and just wants to return a variable back to the “call” point of the function for further uses) | | |
| **Structure** | | | | | |
| struct T  {X1 Y1;  X2 Y2;…} Z1, Z2 | | | define a structure type “T”; X and Y are the members’ ( or called “field”) types and names. Z as the name of the structure variable or structure array (Z or T can be omitted) ([*C++*] the members can be both data and functions, however functions are usually included by class) | | |
| struct T X1, X2 | | | define structure variables, X as the name of variables or structure arrays, T as the name of the structure type defined before ([*C++*] “struct” can be omitted) | | |
| **Class** | | | | | |
| class T  {private:  X1 Y1;  …  public:  X2 Y2; …} Z1, Z2 | | | define a class “T”, X and Y are the members’ types and names; members can be functions; Z as the name of the objects (Z or T can be omitted) (“private”, “public”, and “protected” are called member access specifier) | | |
| T X(Y) | | | make a duplicate of object Y and the new duplicate is X; T is the name of the class. | | |
| T X=Y | | |
| **I/O Stream** | | | | | |
| cout<<X<<Y<<… | | [#include<iostream>] Output X,Y successively in command line of console (special output order -- endl: change to new line;) | | | |
| cin>>X>>Y>>… | | [#include<iostream>] Input X,Y successively in command line of console | | | |
| putchar(X) | | [*C*][*F*][#include<iostream>] output one character X in command line | | | |
| getchar( ) | | [*C*][*F*][#include<iostream>] input one character as the result of this function | | | |
| scanf(X, Y) | | [*C*][*F*] input function | | | |
| printf(X, Y) | | [*C*][*F*] output function | | | |
| **Others** | | | | | |
| exit(X) | | [*F*][#include<stdlib.h>] exit the program, X can be any integer | | | |
| abort() | | [*F*]exit the program, like “exit(X)” | | | |

# Note

|  |  |
| --- | --- |
| Preprocessor  & Compiler  & Macro | 1.以下划线或双下划线开头的名称，多数为编译器定义的变量、宏或类型等。 |
| Function | 1. Functions cannot change the values of real parameters, if the real parameters are from variables. They can only apply the values in their process. If the real parameters are from arrays, they can be changed by function. This is similar to the situation where pointer variables are used. |
| 2. Functions should be declared at the start of the program and out of the main function. Then the functions should be defined after the main function for a clear view. |
| 3. Function Overloading: one can define different functions (different function types, different parameters types and different numbers of parameters) with the same function name, as long as the compiler can identify which function is to apply. |
| 4. Function template can be used in the situation where the same body of function (as well as the number of variables) is needed with different types of parameters. |
| 5. Recursive function: p109 |
| 6. [*C*] The name of a function represents its input address. So one can use the name of a function to assign the pointer of this function, like “function(x,y) => p=function”. ([*C++*] one should use “p=&function”) |
| 7. Using pointers and references as virtual parameters do not need as much space as the usual virtual parameters do. (pointers and references do not copy the real parameters) |
| Array | 1. If using multidimensional arrays as virtual parameters of functions, one can only leave the first dimension undefined. The other dimensions must be explicitly defined. |
| 2. Arrays cannot be assigned as a whole and can only be given values element by element. (because the name of an array is an address) |
| 3. The name of an array represents the address of the first element in that array. This makes it have the same properties as the a pointer variable does, like “p=a” => “p+i=a+i”, “\*(p+i)=\*(a+i)=a[i]=p[i]”. (So the name of an array equals to a pointer variable; the “[i]” sign can also be used by pointer) |
| 4. When an array serves as a virtual parameter in a function, it is processed just like a pointer variable. So when defining a function, one can use either a pointer variable or an array as the virtual parameter for the same purpose (1D arrays correspond to pointer variables pointing to variables; 2D arrays correspond to pointer variables pointing to 1D arrays). However, the name of a “real” array represents a pointer constant and cannot be changed. |
| 5. int a[3][4] => (a[i] is both an element of “a” and a name of an array; a[i]=&a[i])  a==&a[0]; a+1==&a[1];  a[0]=&a[0][0]; a[1]=&a[1][0];  a[0]+1=&a[0][1];  a[0][1]=\*(a[0]+1)=\*(\*(a+0)+1) |
| 6. The character arrays can be assigned using string constant with or without {}. |
| String | 1. String variable can be modified by character, like X[Y] = ‘Z’ is assigning ‘Z’ to No.Y element of X. (that means a string can be regarded as a character array, a pointer of a string can also be regarded as a pointer of a character array) |
| 2. Unlike string constant, string variable does not include ‘\0’ at the end. |
| 3. Like character arrays, string variable can be input/output as a whole. |
| 4. String variable can be operated like basic data types by using “=”, “+”, “><”… |
| 5. String type can be used to define arrays - string arrays. Each element of a string array contains one string variable. |
| 6. Each string variable contains the address of its string constant (without ‘\0’), which means string variables store only the pointers of the strings. |
| 7. “String” type is actually a class defined by C++ standard. |
| Pointer | 1. \*p++ =\*(p++) |
| 2. (\*p)[ ] != \*p[ ]; (the former is a pointer to arrays, the latter is an array of pointer variables) |
| 3. (\*p)(int, int) != \*p(int, int); (the former is a pointer to functions, the latter is an function with the type of pointer) |
| 4. The pointer variables pointing to functions are usually used in C language. |
| Structure | 1. One can include another structure variable as a member in the current structure. OR One can define a pointer variable pointing to variables of a structure in the definition of this structure. |
| 2. The members in the structure can share the same names as other ordinary variables. |
| 3. The initialization of structure variables is similar to that of arrays. (the members are assigned in sequence) |
| 4. The structure variable can be assigned as a whole like an array (structure={x,y,z…} or by element. (“structure.member = …” or “structure.memstructure.member = …”) But the structure variables cannot be output or input as a whole. |
| 5. “&structure” is the address of the first member in that structure. |
| 6. structure.member == (\*p).member == p->member (can also be used with objects of a class) |
| Union | 1. One cannot refer to union variables directly and can only refer to members. |
| 2. All the addresses of a union variable and its members are the same. |
| 3. One cannot initialize a union variable and union variable cannot act as parameters of functions. |
| Class | 1. In the definition of a structure type, all the members are regarded public by default. In the definition of a class, all the members are regarded private by default. (in C++, structure can have virtual function and can be inherited, while it cannot in C. In fact, in C++ it can be replaced by class.) |
| 2. Usually, the function member is declared in the definition of class and defined elsewhere. However, the function member should be added “field qualifier”, like “class::function( )”. |
| 3. All the function members of the same class are stored in the same space, which means the space of an object is only for its data members. |
| 4. Only the public data and functions can be referred to outside the body of class by using “object.member”. |
| 5. The objects from the same class can be assigned as a whole with each other. |
| 6. Classes can be declared like functions before their definition in order to be referred to. However, one cannot use this class to create objects until this class has been defined, because the space for the objects may vary according to the specific definition of the class. This also gives a reason for using references or pointers of this class to declare functions, since they demand constant space.  class ClassName; (declaration of class) |
| 7. One can use a class to declare its function members inside the definition body of that class, which means one can use a class as a data type before it is fully defined (in the definition body). |