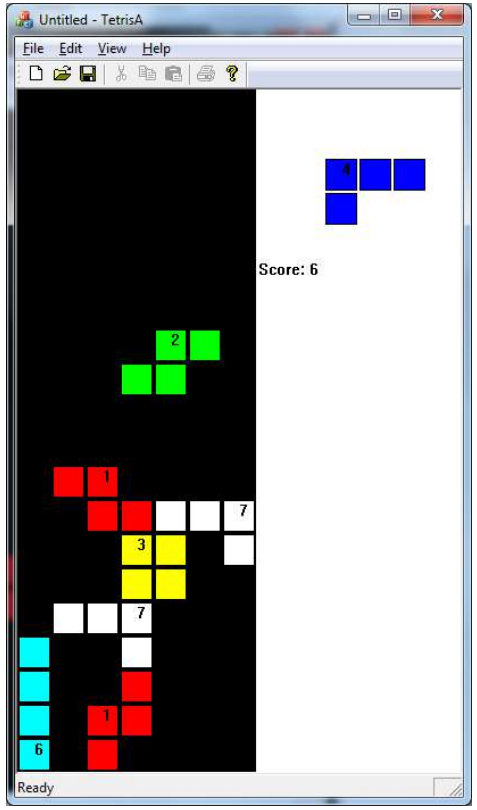
程式設計
(C++ Programming)

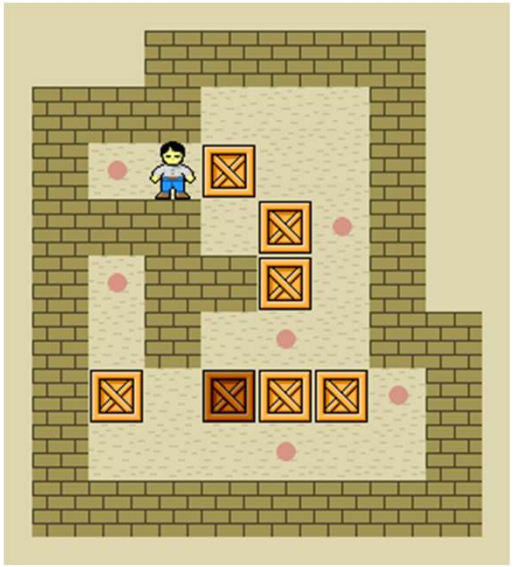
吳坤熹
solomon@mail.ncnu.edu.tw

老師，我已經學過一學期的C語言程式設計了!

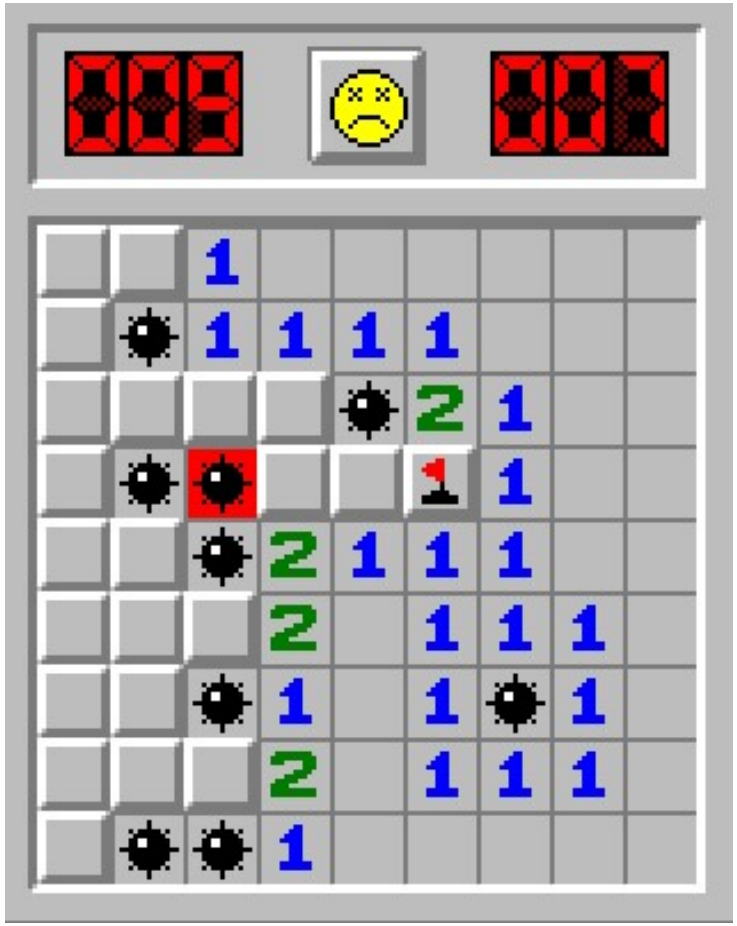
2021 Term Project: TETRIS



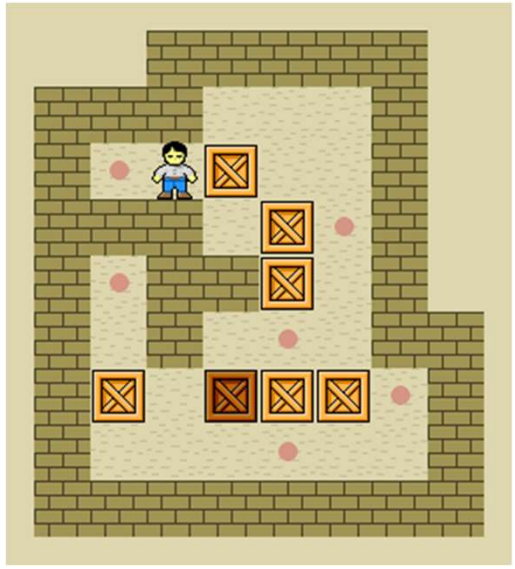
2022 Term Project: Sokoban



2023 Term Project: Minesweeper



2024 Term Project: Sokoban + Network



Grading Criteria

- Participation (10%)
- Homework (10%)
- Quiz (30%)
- Midterm Exam (20%)
- Final Exam (20%)
- Term Project (10%)

Participation

- 老師上課會問問題。你必須盡最大的努力回答。
- 不可以答「不知道！」即使你不知道完整的答案，至少你要講一些話，表示你對這個主題不是一無所知。
- 五秒鐘原則：當聽到問題後，要在五秒鐘內發出有意義的字語。
- 若不確定問題，可試著用自己的話講述一遍，請老師確認。

- 未能回答問題的，視為「人到心未到」。比照缺課辦理。
- 若同學們回答問題的狀況不佳，下週開始就排固定之座位表。依回答問題之狀況計分。
- 請公假之同學，依學校之規定辦理。勿直接 email 給老師或助教。

About Homework

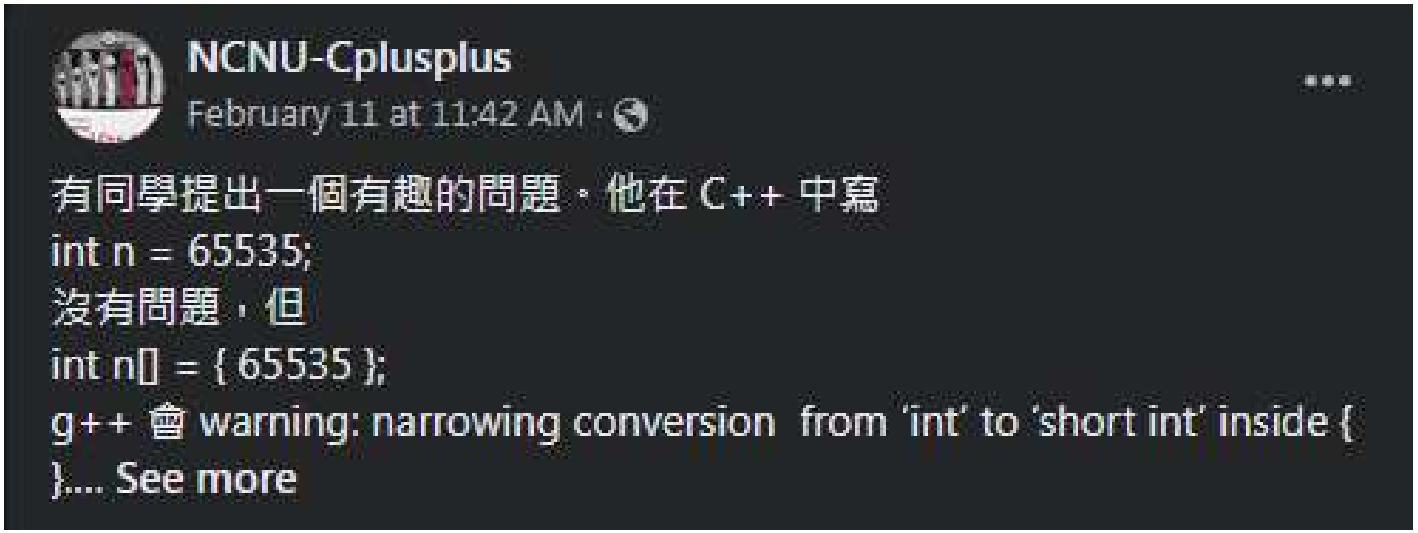
- 遇到問題，可以先問同學，上網查。還是不行，就問學長、助教或老師。
- 不要抄襲，自欺欺人。
  - Teach Students About Plagiarism https://www.educationworld.com/teachers/5-ways-teach-students-about-plagiarism
- 作業有寫有分。
- 遭遇什麼問題，卡在哪裡，就用註解寫在整個檔案的最前面。
- 如果是 Moodle, 就同步註明在作業上傳的文字說明區。

A Few Suggestions

- Take notes 
- Think about it.
- Practice
- Practice again.
- Teach it.
- Identify your weakness and try to improve.

Facebook 討論區: NCNU-Cplusplus

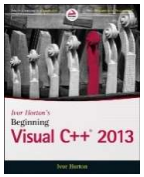
- https://www.facebook.com/NCNU-Cplusplus-716240738432152

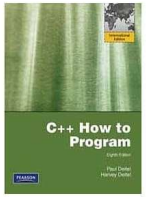


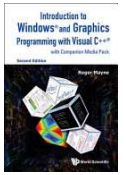
Textbook & References

- 課本：無

- References:
  - Ivor Horton, "Ivor Horton's Beginning Visual C++ 2013", Wrox (May 12, 2013).
  - Paul J. Deitel and Harvey M. Deitel, C++ How to Program, 8th Edition, Pearson Education.
  - Roger Mayne, Introduction to Windows and Graphics Programming with Visual C++ : with Companion Media Pack, 2nd Edition, World Scientific, 2015.
  - [eBook] Mike Dawson, "Beginning C++ game programming", Premier Press, 2004.







2024 Syllabus

- 2/23 Arrays and Pointers
- 3/1 Struct
- 3/8 Macro, Debugger, and Ncurses
- 3/15 Object & Class
- 3/22 Operator Overloading; friend function
- 3/29 Strings, Vectors, Maps, and STL
- 4/5 Spring Vacation
- 4/12 Inheritance, Virtual Function, and Exception Handling
- 4/19 Midterm Exam
- 4/26 MFC Drawing
- 5/3 Keyboard & Menu
- 5/10 Mouse & Drawing
- 5/17 Controls
- 5/24 Multi-thread Programming
- 5/31 Sound File and Audio Output
- 6/7 Special Holiday before Dragon Boat Festival (6/10 Mon)
- 6/14 Final

Today's Topics

- Programming environment & difference between C and C++
- Pointer and array
- Callback function

Your Old C Programs Are Compatible in C++

- C++ retains almost all of C as a subset.
- However, C++ offers better ways to do things as well as some brand-new capabilities.
  - That's what we are going to learn during this semester.

```
#include <stdio.h>

int main()
{
    printf("Hello, NCNU!\n");
    return 0;
}
```

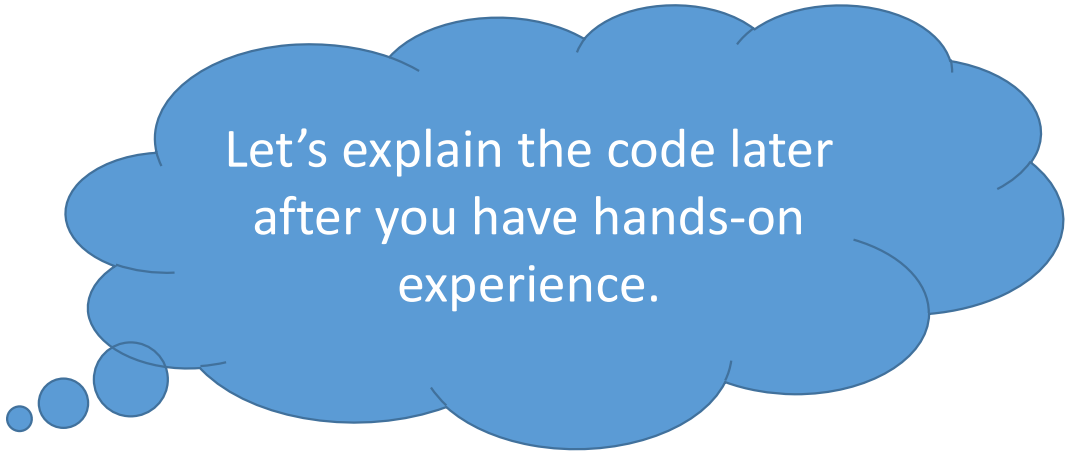
C vs. C++

```c
#include <stdio.h>

int main()
{
    int a, b, c;

    a = 10;
    b = 20;
    c = a + b;
    printf("%d\n", c);

    return 0;
}
```

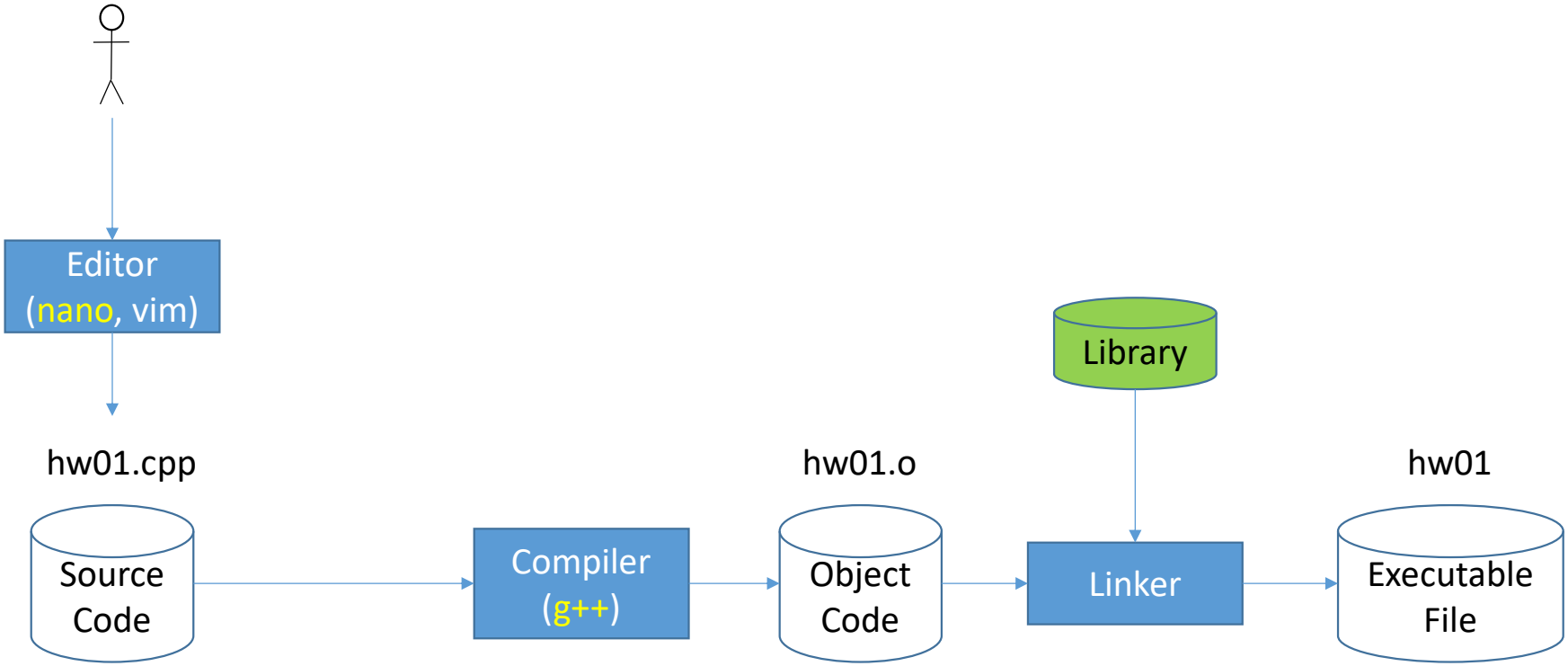
```cpp
#include <iostream>
using std::cout;

int main()
{
    int a, b, c;

    a = 10;
    b = 20;
    c = a + b;
    cout << c << '\n';

    return 0;
}
```

Writing Your First C++ Program

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::out << "Hello NCNU!" << std::endl;
    return 0;
}
```

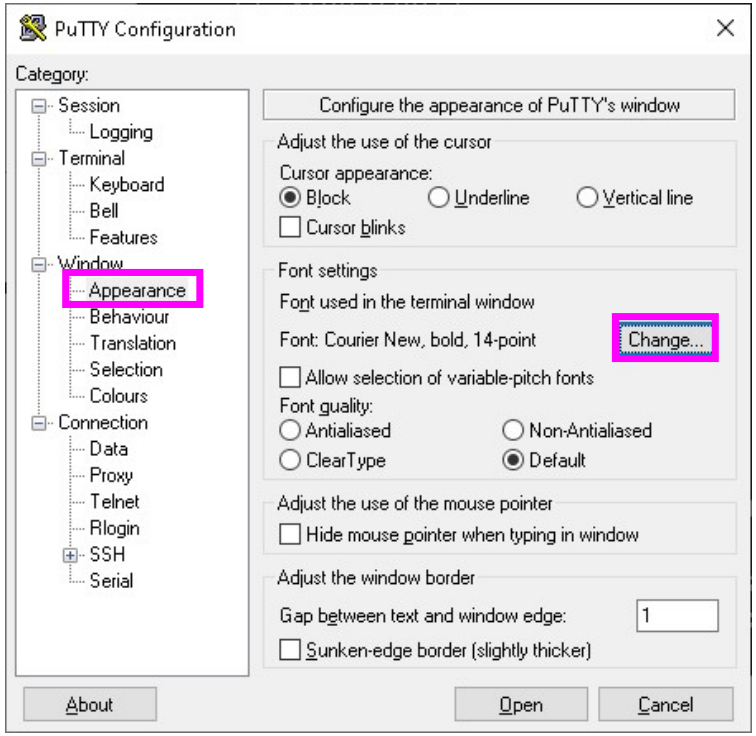


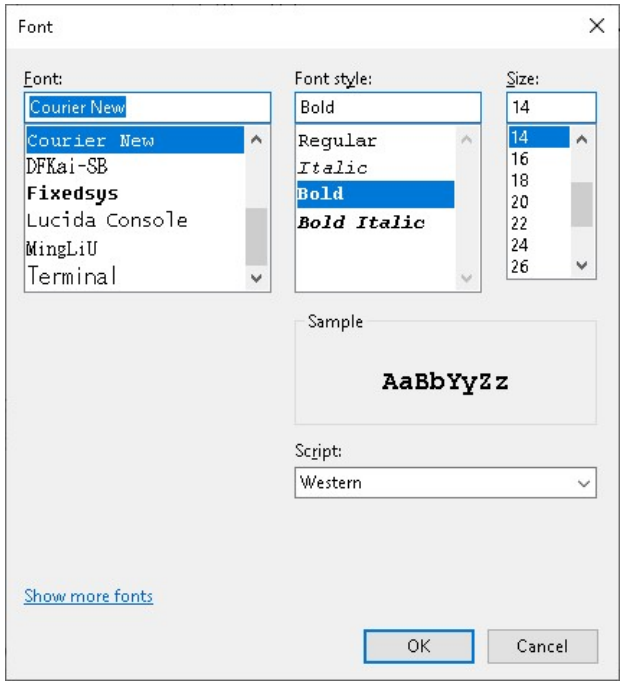
Creation of an Executable File



Hands-On: Create an Executable Program on Lilina

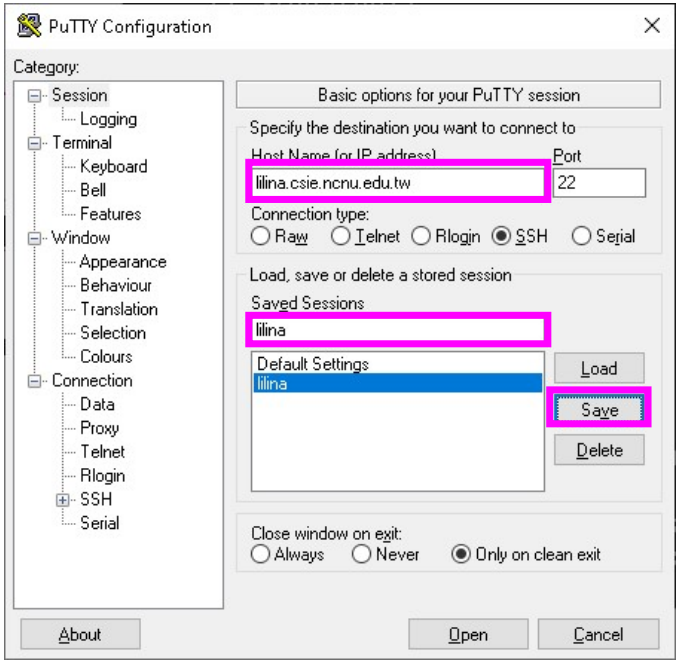
- Use PuTTY to remotely log into lilina.csie.ncnu.edu.tw
  - Change to a larger font, please.





Hands-On: Create an Executable Program on Lilina

- Use PuTTY to remotely log into lilina.csie.ncnu.edu.tw
  - Change to a larger font, please.
  - You may want to save this profile for future access.



Hands-On: Create an Executable Program on Lilina

- Use PuTTY to remotely log into lilina.csie.ncnu.edu.tw
  - Change to a larger font, please.
  - You may want to save this profile for future access.
  - If you don't have an account, please request Prof. Klim Chang (klim@csie.ncnu.edu.tw) to create one for you. You should explain to him that you need this account for this class.
  - If you forget your password, please also request Klim to reset a new password for you. For now, you may temporarily ask your neighbor to login a session for you.

Hands-On: Create an Executable Program on Lilina

- Create a Directory for this class:
  - mkdir CPP # You only do this once.
- Change to the newly created directory
  - cd CPP
- Edit a source file:
  - nano hw01.cpp
  - vim hw01.cpp
- Compile a C++ program (and link with a C++ standard library) to an executable file.
  - g++ hw01.cpp -o hw01
    - You may also name the executable file as “hw01.exe”.
    - If you don’t specify, the default output file name is “a.out”.
- Run the executable file.
  - ./hw01

Comments

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;       // Show a message
    return 0;
}
```

1. Comments are completed ignored by the compiler. They are for humans.
2. They can help programmers (including yourself) understand your intentions.
3. Anything after "//" in the line will be ignored.
4. You may also use the old *C-style* comments (/* … */), which can span multiple lines.

Whitespace

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

1. C++ compilers ignore blank lines.
   - Actually, all whitespace (spaces, tabs, and new lines) are ignored.
2. Using blank lines to separate sections of code can make your programs clearer.

newline

space

tab

Bad Example:

- The following code is legal, but difficult to read:

```
#include <iostream>
int main() { std::cout << "Hello NCNU!" << std::endl; return 0; }
```

Including Other Files

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

1. A line beginning with the # symbol is a preprocessor directive.
2. The *preprocessor* runs before the compiler. It substitutes text based on these directives.
3. #define will define a macro for further use.
4. #include will include the contents of another file.

#define

```
g++ -E a.cpp -o a.i
```

```
#define N       10

int main()
{
    printf("%d\n", N);
    return 0;
}
```

```
int main()
{
    printf("%d\n", 10);
    return 0;
}
```

#include

```
g++ -E a.cpp | grep -v '^#'
```

- a.cpp

```
int main()
{
    printf("%d\n", N);

#include "a.file"
```

- a.file

```
    return 0;
}
```



```
int main()
{
    printf("%d\n", N);

    return 0;
}
```

Including Other Files

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

1. iostream is called a *header file*.
   - Normally, it contains function headers. (Actually, function prototypes.)
2. It is part of the standard library, so it is surrounded with < and >.
3. On lilina, you can find it at /usr/include/c++/5.5.0/iostream
4. In the previous example, the included filename is surrounded by a pair of double quotes ("), so the compiler will search it in the current directory..

Function Prototype

```
int add(int a, int b)
{
   return a+b;
}
```

```
int add(int a, int b) ;
```

main()

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

1. A function is a group of code that can do some work and (usually) return a value.
2. A C++ programs must have a function called main().
3. The execution starts from main().

Every statement ends with a semicolon(;)

Code between two curly braces is called a block and is usually indented.

Displaying Text

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

1. `cout` is an object, defined in the file `iostream`.
2. `cout` is used to send data to the standard output stream.
   - Normally, this is the computer screen.
3. It can handle many different data types: int, char, and even expressions.
   - `std::cout << a << '+' << b << '=' << a+b;`

end-of-line: This will send a newline character ('\n') to output.

Displaying Text

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

`std` is a namespace
You prefix a namespace using the *scope resolution operator* (::).

namespace

- When multiple programmers work in a project, it is quite often that they choose the same function names or global variable names in their code:

undergraduate.h

```
#include <math.h>

const int THRESHOLD_TO_PASS = 60;

int adjust(int n)
{
    return (int) sqrt(n) * 10;
}
```

graduate.h

```
const int THRESHOLD_TO_PASS = 70;

int adjust(int n)
{
    return (int) n * 0.7 + 70;
}
```

```
#define N      10
#include "undergraduate.h"
#include "graduate.h"

int main()
{
   int grade[] = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81};
   for (int i=0; i<N; ++i)
      grade[i] = adjust(grade[i]);
   return 0;
}
```

error: redefinition of ‘int adjust(int)’

namespace

- You may certainly ask the two programmers to modify their code.
- Or,

```
#include <iostream>
#define N       10

namespace Undergraduate
{ #include "undergraduate.h" }

namespace Graduate
{ #include "graduate.h" }

int main()
{
    int grade[] = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81};
    for (int i=0; i<N; ++i)
        std::cout << Undergraduate::adjust(grade[i]) << std::endl;
    std::cout << Graduate::THRESHOLD_TO_PASS << std::endl;
    return 0;
}
```

1. namespace is a convenient mechanism in C++ to prevent name collisions (functions, variable names).
2. All C++ standard library types and functions are declared in the `std` namespace.
3. For more information about namespace, see https://docs.microsoft.com/en-us/cpp/cpp/namespaces-cpp?view=msvc-170

Employing a `using` Directive

- This allows all the names in a namespace to be used without the namespace-name as an explicit qualifier.

```
// Hello NCNU
// A first C++ program

#include <iostream>


int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

```
// Hello NCNU 2.0
// Demonstrates a using directive

#include <iostream>
using namespace std;

int main()
{
    cout << "Hello NCNU!" << endl;
    return 0;
}
```

Employing `using` Declarations

- You may also write two using declarations to precisely specify which elements from the `std` namespace you want to use in your program.

```
// Hello NCNU 2.0
// Demonstrates a using directive

#include <iostream>
using namespace std;


int main()
{
    cout << "Hello NCNU!" << endl;
    return 0;
}
```

```
// Hello NCNU 3.0
// Demonstrates using declarations

#include <iostream>
using std::cout;
using std::endl;

int main()
{
    cout << "Hello NCNU!" << endl;
    return 0;
}
```

Some textbooks prefer this one, because it looks simpler (only one line).
However, if you always import everything from the namespace, why did you give it an independent namespace?

Q: “using namespace std” is a bad habit?

- Consider “solomon.h” :

```
#include <stdio.h>
void cin()  { printf("This is my cin.\n"); }
void cout() { printf("This is my cout.\n"); }
```

- This works fine.
- However, if you think it’s troublesome to prefix “std::”, and add an instruction “using namespace std;” ...

- In my main program:

```
#include <iostream>
#include "solomon.h"

int main() {
    std::cout << "Hello\n";
    cin();
    return 0;
}
```

using namespace std;

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

int main()
{
   std::cout << "Hello\n";
   cin();
   return 0;
}
```

Adding this line will cause syntax error:
error: reference to ‘cin’ is ambiguous

Assign Your Library a Namespace

```
#include <iostream>
namespace Solomon {
#include "solomon.h"
}

int main()
{
    std::cout << "Hello\n";
    Solomon::cin();
    return 0;
}
```

This is better, because you don't need to worry whether the functions or classes defined in your library will conflict with something that was also defined in the default library. (There are many, and you learned less than 1%.)

This is the so-called "Scope Resolution Operator" (::).

“using” can save your trouble in prefixing with scope resolution operators.

```
#include <iostream>
namespace Solomon {
#include "solomon.h"
}
using std::cout;
using Solomon::cin;

int main()
{
    cout << "Hello\n";
    cin();
    return 0;
}
```

“using namespace Solomon;” seems convenient to import all declarations

```
#include <iostream>
namespace Solomon {
#include "solomon.h"
}
using namespace Solomon;

int main()
{
    std::cout << "Hello\n";
    cin();
    return 0;
}
```

But that works if you only retrieve one library

```
#include <iostream>
namespace Solomon {
#include "solomon.h"
}
using namespace std;
using namespace Solomon;

int main()
{
    std::cout << "Hello\n";
    cin();
    return 0;
}
```

error: reference to ‘cin’ is ambiguous

Returning a Value from main()

```
// Hello NCNU
// A first C++ program

#include <iostream>

int main()
{
    std::cout << "Hello NCNU!" << std::endl;
    return 0;
}
```

- Your program returns an integer to the operating system to indicate whether the program runs successfully.
- 0 implies “successful”.

```
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

int main()
{
    int n;
    int ret_value;
    cin >> n;
    if (n % 2 == 0) {
        cout << "divided by 2." << endl;
        ret_value = 0;
    } else {
        cout << "Cannot be divided by 2." << endl;
        ret_value = -1;
    }

    return ret_value;
}
```

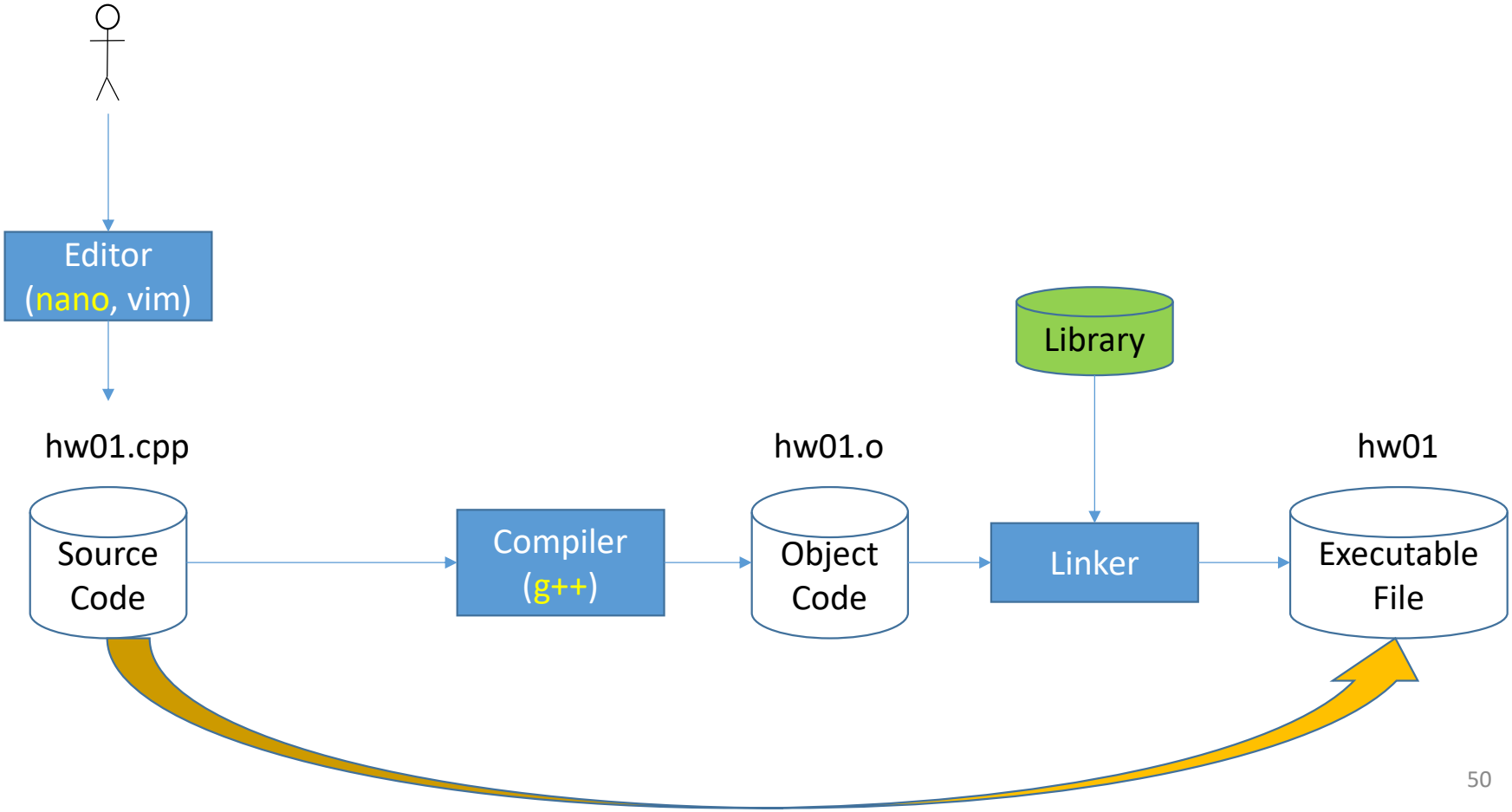
Status Code ($?)

PS1="\!> " # prompt string
1> a.out
6
divided by 2.
2> echo $?
0
3> a.out
3
Cannot be divided by 2.
4> echo $?
255
5> a.out && date
6
divided by 2.
Sat Feb 5 15:26:01 CST 2022
6> a.out && date
3
Cannot be divided by 2.

If a.out succeeds, run “date”.

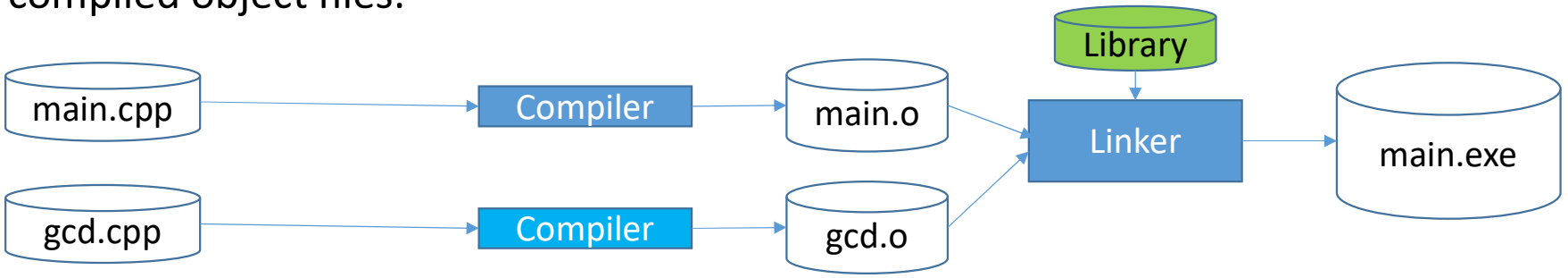
推薦課程：
張克寧老師
“Unix使用入門”

Directly from Source Code to Executable?



Why Separate Compiling and Linking?

- When your program gets more complex, it may consist of many .cpp files. If you only modify one of them, you don't need to compile every .cpp file.
  - You only need to compile the one which is modified, and link it with the remaining compiled object files.



- Sometimes you only get a (compiled) object file, without the source code.
- Some functions may be developed by a different language, and must be compile with a different compiler.

main.cpp

```
#include <iostream>
using std::cout;
using std::endl;

int gcd(int a, int b);              // function prototype

int main()
{
    cout << gcd(36, 24) << endl;
    return 0;
}
```

Errors and Warnings

- Compile errors.
  - This is also called "syntax errors".
  - As a result, an object file is not produced.
  - At the beginning you may mostly be stuck by this kind of errors. However, you will find that they are the easiest ones to handle, because the compiler can detect them for you.
  - Compilers may issue "warnings", too. Although an object file is still generated, you should inspect the reasons which trigger the warning, and fix them.
- Link errors.
  - This indicate that something your program references externally cannot be found.
  - These errors are usually solved by adjusting the referenced function names or linking the right library.

```
g++ main.cpp
/tmp/ccPYOFxb.o: In function `main':
main.cpp:(.text+0xf): undefined reference to `gcd(int, int)'
collect2: error: ld returned 1 exit status
```

- Run-time errors.
  - Even if your program is correct, it can crash existing file.
- Logical errors.
  - These are the most difficult errors to find. Your program can run without crashing, but the result is wrong!
  - A good debugging tool may be helpful to identify the statements which cause the bug.

Hands-On: Create an Executable Program on Lilina

- Edit a source file:
  - nano main.cpp
  - vi main.cpp
- Compile a C++ program to an object file:
  - g++ -c main.cpp
- Link the object file (with a function developed by solomon) to an executable file.
  - g++ main.o ~solomon/CPP/gcd.o -o main.exe
- List files in current working directory.
  - ls

Fundamental Data Types in C++

Type	Size	Values
bool		true or false
char	1 byte	256 character values
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
int	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	0 to 4,294,967,295
float	4 bytes	3.4E +/- 38 (7 significant digits)
double	8 bytes	1.7E +/- 308 (15 significant digits)

- Q: What happens if you add 1 to a `short` with value 32767?
- A: It "wraps around" and becomes -32768.

1. C++ requires very variable to be declared before it is used.
2. This allows you (and the compiler) to know how much memory space is needed to run a function.

Naming Variables

- There are only a few rules to name a variable or a function:
  - An identifier can contain only numbers, letters, and underscores.
  - An identifier cannot start with a number.
    - Prevent using identifiers which starts with an underscore.
  - An identifier cannot be a C++ keyword.
- Here are some guidelines for choose good variable names:
  - Choose descriptive names. For example, score is better than s. (One exception involves variables used for a short period. In that case, single-letter variable names are appropriate.)
  - Be consistent. Is it `high_score` or `highScore`? Many Python programmers prefer `highScore` (camel case) while C programmers prefers `high_score`. You may have our own style as long as you are consistent.
  - Follow the traditions of the language. For example, C++ variable names start with a lowercase letter, while constants are all uppercase letters.
  - Avoid using an underscore as the first character of your identifiers. Names that begin with an underscore usually have special meaning.

Declaring and Initializing Variables

```
// game_stats.cpp
// Game Stats
// Demonstrates declaring and initializing variables

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
    int score;
    double distance;
    char playAgain;
    bool shieldsUp;

    short lives, aliensKilled;

    score = 0;
    distance = 1200.76;
    playAgain = 'y';
    shieldsUp = true;
    lives = 3;
    aliensKilled = 10;

    double engineTemp = 6572.89;

    cout << "\nscore: "        << score        << endl;
    cout << "distance: "       << distance     << endl;
    cout << "playAgain: "      << playAgain    << endl;
    // skip shieldsUp since you don't generally print Boolean values
    cout << "lives: "          << lives        << endl;
    cout << "aliensKilled "    << aliensKilled << endl;
    cout << "engineTemp: "     << engineTemp << endl;

    int fuel;
    cout << "\nHow much fuel? ";
    cin >> fuel;
    cout << "fuel: " << fuel << endl;

    typedef unsigned short ushort;
    ushort bonus = 10;
    cout << "\nbonus: " << bonus << endl;

    return 0;
}
```

Declaring variables before you use them.

Assigning values to variables

Declaring and Initializing Variables

```
// game_stats.cpp
// Game Stats
// Demonstrates declaring and initializing variables

#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
    int score;
    double distance;
    char playAgain;
    bool shieldsUp;

    short lives, aliensKilled;

    score = 0;
    distance = 1200.76;
    playAgain = 'y';
    shieldsUp = true;
    lives = 3;
    aliensKilled = 10;
    double engineTemp = 6572.89;

    cout << "\nscore: "        << score        << endl;
    cout << "distance: "      << distance     << endl;
    cout << "playAgain: "     << playAgain    << endl;
    // skip shieldsUp since you don't generally print Boolean values
    cout << "lives: "         << lives        << endl;
    cout << "aliensKilled "   << aliensKilled << endl;
    cout << "engineTemp: "    << engineTemp << endl;

    int fuel;
    cout << "\nHow much fuel? ";
    cin >> fuel;
    cout << "fuel: " << fuel << endl;

    typedef unsigned short ushort;
    ushort bonus = 10;
    cout << "\nbonus: " << bonus << endl;

    return 0;
}
```

Getting User Input

Use `typedef` to create shorter names for types.

Using Constants

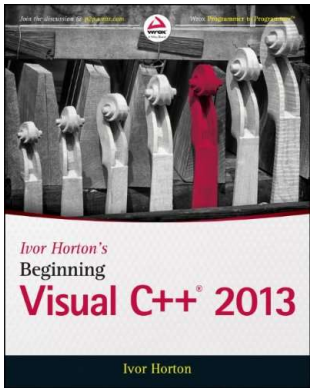
```
// game_stats3.cpp

#include <iostream>
using std::cout;
using std::endl;

int main() {
    const int ALIEN_POINTS = 150;
    int alienKilled = 10;
    int score = alienKilled  * ALIEN_POINTS;
    cout << "score: " << score << endl;

    enum Difficulty {NOVICE, EASY, NORMAL, HARD, UNBEATABLE};
    Difficulty myLevel = EASY;

    enum ShipCost {FIGHTER_COST = 25, BOMBER_COST, CRUISER_COST = 50};
    ShipCost myCost = BOMBER_COST;
    cout << "To upgrade my ship to a Cruiser will cost "
        << (CRUISER_COST - myCost) << " points.\n";
    return 0;
}
```

1. You cannot assign a new value to a constant. It'll generate a compile error.
2. Conventionally, constant names are in all uppercase letters.
3. Constants make programs clearer. If you have many "magic numbers" in your program like `score = score + 150`, you may not know what the value represent.
4. Constants also make changes easy. If you decide that each alien should be worth 250 points, you only need to change the initialization of ALIEN_POINTS.

Using Enumerations

```
// game_stats3.cpp

#include <iostream>
using std::cout;
using std::endl;

int main() {
    const int ALIEN_POINTS = 150;
    int alienKilled = 10;
    int score = alienKilled  * ALIEN_POINTS;
    cout << "score: " << score << endl;

    enum Difficulty {NOVICE, EASY, NORMAL, HARD, UNBEATABLE};
    Difficulty myLevel = EASY;

    enum ShipCost {FIGHTER_COST = 25, BOMBER_COST, CRUISER_COST = 50};
    ShipCost myCost = BOMBER_COST;
    cout << "To upgrade my ship to a Cruiser will cost "
        << (CRUISER_COST - myCost) << " points.\n";
    return 0;
}
```

1. An enumeration is a set of `unsigned int` constants, called *enumerators*.
2. By the default, the value of enumerators begins at 0 and increases by 1.
3. Note: Although the value of BOMBER_COST is 26,
   `myCost = 26`
   is not valid. C++ only allows
   `myCost = BOMBER_COST`

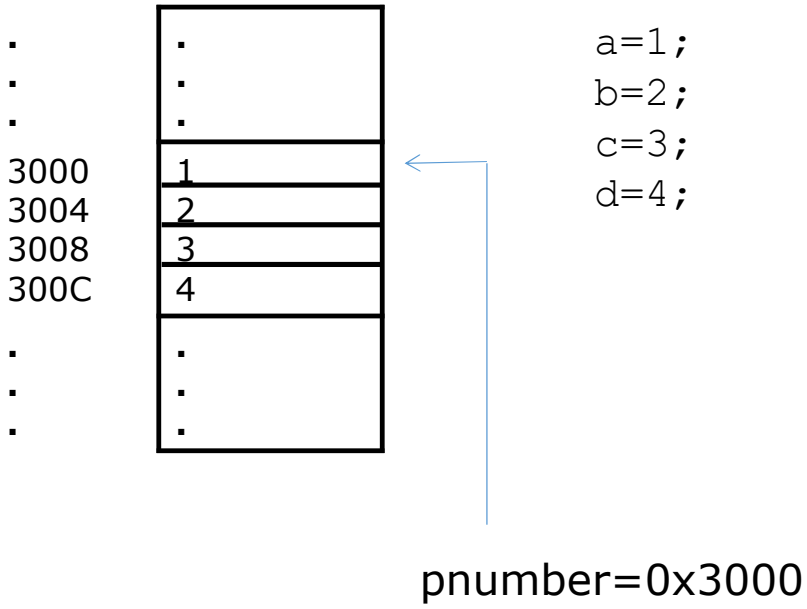


Chapter 4

Pointers

Indirect Data Access with Pointers (P.146)

- Each memory location which you use to store a data value has an address.
- A pointer is a variable that stores an address of another variable (of a particular type).
  - e.g., the variable **`pnumber`** is a pointer
  - It contains the address of a variable of type `int`
  - We say **`pnumber`** is of type 'pointer to `int`'.



The Address-Of Operator

- How do you obtain the address of a variable?
  - `pnumber = &number;`
    - Store address of `number` in `pnumber`

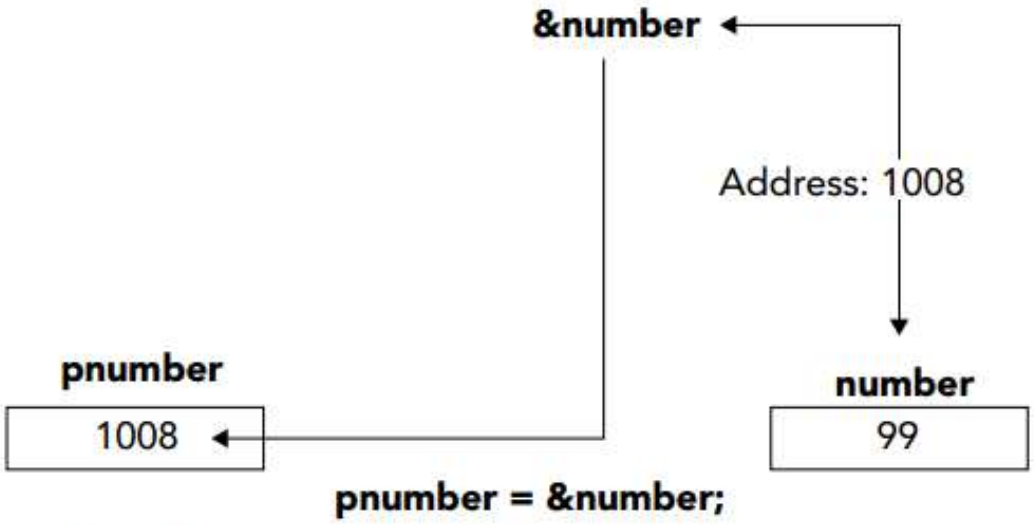


FIGURE 4-4 (P.147)

Show the memory address

```
#include <iostream>
using std::cout;
using std::endl;
using std::hex;

int main()
{
    int a=1, b=2;

    cout << hex << &a << hex << &b << endl;
    printf("%x %x\n", &a, &b);
    printf("%d %d\n", &a, &b);

    return 0;

}
```

```
0x7fffffffdafc0x7fffffffdaf8
ffffdafc ffffdaf8
-9476 -9480
```

The Indirection Operator

- Use the indirection operator *, with a pointer to access the contents of the variable that it points to.
  - Also called the “de-reference operator”

- Ex4_06.cpp on P.149
  - *pnumber += 11;
  - number1 += 11;
  - number1 = number1 + 11;

De-reference a Pointer with Arithmetic

- Assume `pdata` is pointing to `data[2]`,
  - `*(pdata + 1) = *(pdata + 2);`

  is equivalent to
  - `data[3] = data[4];`

Tips: *(data + i) == data[i]

double data[5];

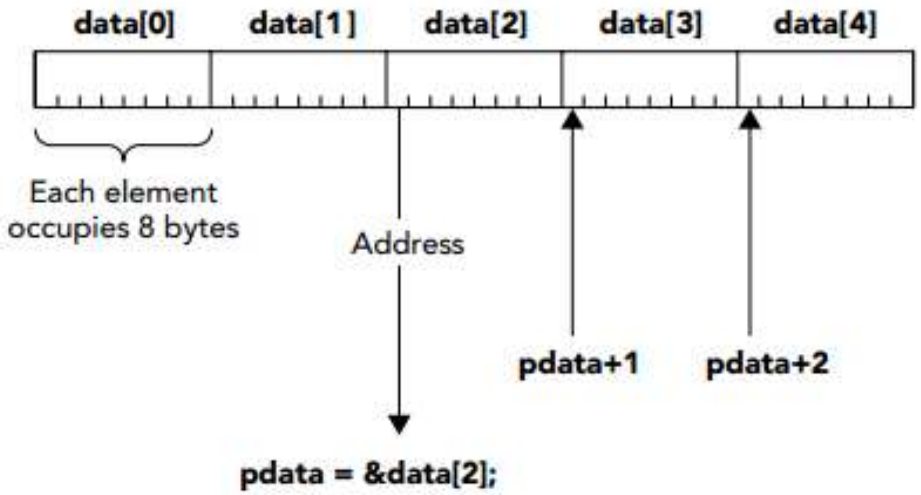


FIGURE 4-7 (P.157)

Free Store (Heap)

- To hold a string entered by the user, there is no way you can know in advance how large this string could be.

- Free Store - When your program is executed, there is unused memory in your computer.
- You can dynamically allocate space within the free store for a new variable.

The new Operator

- Request memory for a double variable, and return the address of the space
  - `double* pvalue = NULL;`
  - `pvalue = new double;`
- Initialize a variable created by new
  - `pvalue = new double(9999.0);`
- Use this pointer to reference the variable (indirection operator)
  - `*pvalue = 1234.0;`

pvalue = (double*) malloc(8);
*p = 9999.0;

The delete Operator

- When you no longer need the (dynamically allocated) variable, you can free up the memory space.
  - `delete pvalue;`
    - Release memory pointed to by pvalue
  - `pvalue = 0;`
    - Reset the pointer to 0

- After you release the space, the memory can be used to store a different variable later.

Allocating Memory Dynamically for Arrays

- Allocate a string of twenty characters
  - `char* pstr;`
  - `pstr = new char[20];`
  - `delete [] pstr;`
    - Note the use of square brackets to indicate that you are deleting an array.
  - `pstr = 0;`
    - Set pointer to null

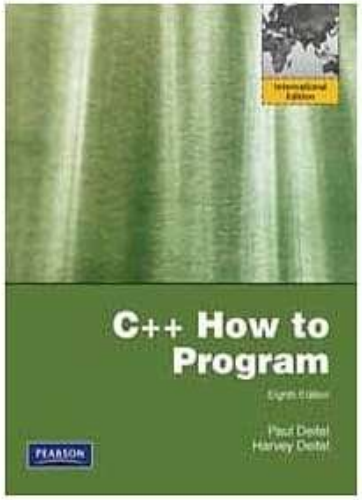
Type Conversion

```
// C
double f = 3.14;
int a = (int) f;

int* p = (int*) &f ;
```

```
// C++
double f = 3.14;
int a = static_cast<int>(f);

int* p = reinterpret_cast<int*>( &f );
```

memory.cpp

Chapter 8
Pointers



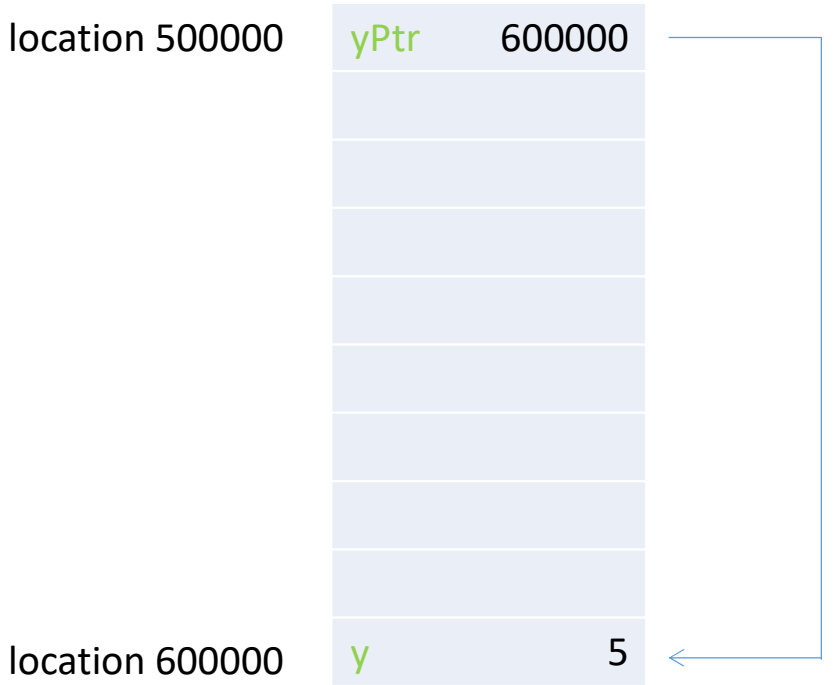
Paul J. Deitel and Harvey M. Deitel,

C++ How to Program,

8th Edition, Pearson Education.

Address-of Operator (&) (P.364)
Dereference Operator (*)

```
int y = 5;
int* yPtr = &y;

*yPtr = *yPtr + 1;
// *&y = *&y + 1;
// y = y + 1;
```



Q: What will the following code display?

```
int c = 'A';
int* pc = &c;
cout << *pc;
```

Pass-by-Value vs. Pass Pointers to a Function

```
void inc(int n)
{
    n += 10;
    cout << n << endl;
}

int main()
{
    int n = 10;
    inc(n);
    cout << n << endl;
    return 0;
}
```

```
20
10
```

```
void inc(int* n)
{
    *n += 10;
    cout << *n << endl;
}

int main()
{
    int n = 10;
    inc(&n);
    cout << n << endl;
    return 0;
}
```

```
20
20
```

C++ Supports Pass-by-Reference

```
void inc(int& n)
{
    n += 10;
    cout << n << endl;
}

int main()
{
    int n = 10;
    inc(n);
    cout << n << endl;
    return 0;
}
```

```
20
20
```

```
void inc(int* n)
{
    *n += 10;
    cout << *n << endl;
}

int main()
{
    int n = 10;
    inc(&n);
    cout << n << endl;
    return 0;
}
```

```
20
20
```

8.5 Using `const` with Pointers (P.371)

```
#include <iostream>

void show(char* s)
{ std::cout << s << std::endl; }

int main()
{
    show("Good morning.");
    return 0;
}
```

Warning: conversion from string literal to 'char *' is deprecated

Implicit conversion from ‘char*’ to ‘const char*’ is all right.

Four ways to pass a pointer to a function (P.372)

- Non-constant Pointer to Non-constant Data
  - `int* pNumber`
- Non-constant Pointer to Constant Data
  - `const int* sizePtr`
- Constant Pointer to Non-Constant Data
  - `int* const pArray`
- Constant Pointer to Constant Data
  - `const int* const pReadOnlyArray`

reinterpret_cast (P.383)

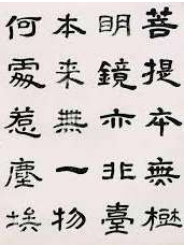
- int* and char* are different data types.
  - Although both are a memory address
    - int* points to an integer (4 bytes)
    - char* points to a character (1 byte)
  - when dereferenced, these two pointers retrieve different data.
- Consider the code
  - char c = 65;
    int* pc = &c;
  - Error: cannot convert ‘char*’ to ‘int*’

- int* pc = reinterpret_cast<int*>(&c);

pointer to void (`void*`)

- `void*` is a generic pointer capable of representing any pointer type.
- All pointer types can be assigned to a pointer of type `void*` without casting.
- However, a pointer of type `void*` cannot be assigned directly to a pointer of another type. It must be cast.
  - `char a[4] = { 'A' };`
  - `void* pa = a;`
  - `int* p = reinterpret_cast<int*>(pa); // (int*) pa`
  - `cout << reinterpret_cast<int*>( a );`

Don’t Mix Up `void` and `void*`



- `void f(int n);`
  - The function returns nothing.

無

- `void* g(int n);`
  - The function returns a pointer, which may point to anything.

空

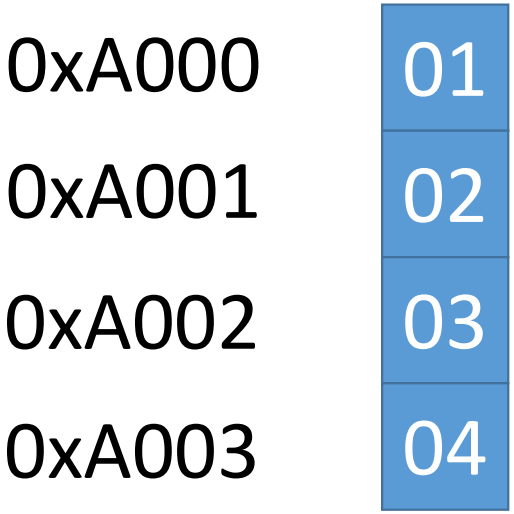
It should be named as “generic*”

空明拳是周伯通被黃藥師困於桃花島時，在所住山洞裡自創的武功。
空明拳共有七十二路，第一路空碗盛飯、第二路空屋住人。
一只碗只因為中間是空的，才有盛飯的功用，倘若它是實心的一塊瓷土，還能裝甚麼飯？
建造房屋，開設門窗，只因為有了四壁中間的空隙，房子才能住人。倘若房屋是實心的，
磚頭木材四四方方的砌上這麼一大堆，那就一點用處也沒有了。

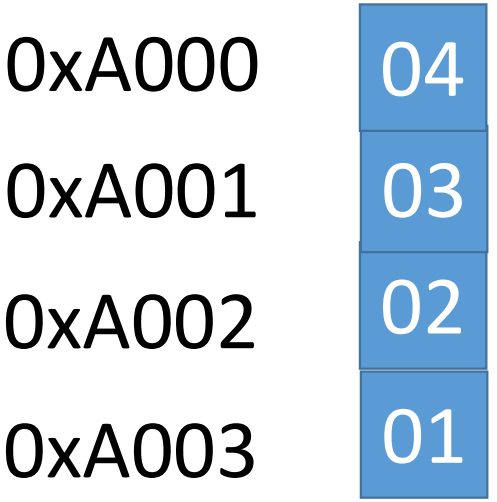
How to Store a 4-Byte Integers in Memory

- 16909060_{10} = 0x01020304 (01020304_{16})
- How will it be stored in memory?

Big-Endian



Little-Endian



Exercise: Big-Endian vs. Little Endian

- Big Endian
  - IBM S/390, Motorola 68k, Sun SPARC,
  - "Big Endian is Effectively Dead"
- Little Endian
  - Intel x86
- Bi-Endian
  - PowerPC, ARM

On your host, can you write a program to inspect whether it stores integers in big-endian or little-endian?

integer 16909060 is stored at

Using pointer ...

0093FAB8	4
0093FAB9	3
0093FABA	2
0093FABB	1

Using union ...

0093FA7C	4
0093FA7D	3
0093FA7E	2
0093FA7F	1

8.12 Function Pointers (P.390)

- When you encounter the terminology "callback function" in the future, come back to study this concept again.
- Passing a "pointer to function" into a function makes your program flexible.

```
int add(int a, int b)
{ return a + b; }

int multiply(int a, int b)
{ return a * b; }

int main()
{
    int (*f)(int, int);

    f = add;
    cout << f(12, 3) << endl;

    f = multiply;
    cout << f(12, 3) << endl;
}
```

The parenthesis cannot be omitted.

An Array of Function Pointers

```
int add(int a, int b)
{ return a + b; }

int sub(int a, int b)
{ return a - b; }

int multiply(int a, int b)
{ return a * b; }

int divide(int a, int b)
{ return a / b; }

int main()
{
    int (*f[4])(int, int) = { add, sub, multiply, divide };

    for (int i=0; i<4; ++i)
        cout << f[i](12, 3) << endl;

    return 0;
}
```

Callback Function

```
void show( int a, int b, int (*f)(int, int) )
{ cout << f(a, b) << endl; }

int add(int a, int b)
{ return a + b; }

int sub(int a, int b)
{ return a - b; }

int multiply(int a, int b)
{ return a * b; }

int divide(int a, int b)
{ return a / b; }

int main()
{
    int (*f[4])(int, int) = { add, sub, multiply, divide };

    for (int i=0; i<4; ++i)
        show(12, 3, f[i]);

    return 0;
}
```

qsort(void *base, size_t nmemb, size_t size,
int (*compar)(const void *, const void *));

```
#include <iostream>

int cmp(const void* p1, const void *p2) {
    int n1 = *reinterpret_cast<const int*>(p1);
    int n2 = *reinterpret_cast<const int*>(p2);
    return n1 - n2;
}

int main() {
    int a[] = { 1, 3, 5, 7, 2, 4, 6 };
    for (size_t i=0; i< sizeof(a)/sizeof(a[0]); ++i) std::cout << a[i] << ' ';
    std::cout << '\n';

    qsort(a, sizeof(a)/sizeof(a[0]), sizeof(a[0]), cmp);

    for (size_t i=0; i< sizeof(a)/sizeof(a[0]); ++i) std::cout << a[i] << ' ';
    std::cout << '\n';
    return 0;
}
```

Why Use Pointers? (P.148)

- Use pointer notation to operate on data stored in an array
  - data[i] = *(data + i)
- Allocate space for variables dynamically.
  - new & delete
- Enable access within a function to arrays, that are defined outside the function
  - swap(&a, &b)
  - This is not so important in C++, which support pass-by-reference.
- Refer to different functions with a pointer
  - Function Pointers (Callback Functions)

請這幾位同學下課後來找我

- 111321065 孫玉玲
- 112321019 藍俊翔
- 112321079 黃亮穎
- 112321080 林玉文
- 112321015 郭品鑫