Clean and Manageable Code

- Compile cleanly
  - No warnings (gcc –Wall)

- “Makefile” is required for each project
  - Eclipse, Visual Studio generate their own
  - But you need to know how to write a Makefile

- Version control using “subversion”
  - Optional, but helpful to keep multiple versions of source code
Code Manage

- **Re-Read code**
  - Pair programming helps
  - Write a clean code for other people to review

- **Correctness**
  - Correct is better than fast
  - Simple is better than complex – prefer clarity over cuteness
  - Write code for people first, then machines

  - “Premature optimization is the root of all evil”
  - “It is far easier to make a correct program fast than to make a fast program correct”
C++ Coding Style

- Minimize global variables
  - Do not use “static” for local variables until necessary
  - Information hiding: do not expose internal information from an object that provides an abstraction

- Use “const” proactively (avoid #define)
  - Allows better type checking and reduces name space pollution

- Avoid macros & magic numbers
  - Bad examples
    ```cpp
    #define MAX(a,b) ((a)>(b)?(a):(b))
    MAX(a++, b)     MAX(a++, b+1)
    ```
    ```cpp
    #define NEXT(x) ((x)++)
    for (i=0; i<100; NEXT(i))
    {
      ...
    }
    ```
C++ Coding Style

- Prefer compile and link time errors to run time errors
  - Strong type check, eliminating all warnings help

- Declare variables as locally as possible
  - Less pollution (side-effect) during function calls

- Always initialize variables

- Avoid a long functions
  - Max lines per function is about 25 lines
Declaration vs. Definition

- **Declaration**
  - `extern int i;
  - int add (int, int);

- **Definition**
  - `int i; // also declaration`
  - `int add (int x, int y) {
      return x + y;
   }`

```cpp
#include <iostream>
using namespace std;

extern int i;
int add (int, int);

int main ()
{
    int a;
    cin >> a;
    a = add(a, i);
    cout << a << endl;
}

int add (int x, int y)
{
    return x + y;
}
```
Initialization

- **Direct initialization**
  - Initialized when a variable is created by giving its initial value
  - More efficient for class objects

```cpp
#include <string>
float phi (3.14);
std::string ten_As (10, 'A');
Point p (1,1);
```

- **Copy initialization**
  - Assigned by replacing the current value with a new one

```cpp
float phi = 3.14;
std::string ten_As = "AAAAAAAAAAA";
Point p;
p.x = 1;
p.y = 1;
```
String Type

- C++ library adds **string** type (class)
- Null terminated char array

```cpp
#include <iostream>
#include <string>
using namespace std;

char charr1[20] = "data_structure";  
string str1 = "software"; 
char charr2[20];  
string str2;  
charr2 = charr1;  // ERROR (compile-time)  
str2 = str1;  // OK  
str2 = str1 + str2;  // concatenation  
cin >> charr1;  // Can cause an ERROR  
cin >> str1;  // OK
```
Avoid macro and magic number

```c
#define N 100
for (i=0; i < N; i++)
{
    ...
}
```

→

```c
const int bufSize = 100;
for (i=0; i < bufSize; i++)
{
    ...
}
bufSize = 500; // ERROR
```

Const variables accept both const and non-const

Non-const variables accept only non-const

```c
int age = 20;
const int* pt = &age;
*pt = 30; // ERROR
age = 30; // OK
```

```c
const int age = 20;
int* pt = &age; // ERROR
*pt = 30;       // Otherwise,
                // age can be changed
```
### Const Pointer Type

- **Pointer to const int**
  - Pointer itself is not a constant, but the location is

  ```c
  int age = 20;
  const int* pt = &age;

  int sage = 80;
  pt = &sage;  // OK to point to another
  *pt = 40;    // ERROR
  ```

- **Const pointer to int**
  - Pointer itself is a constant

  ```c
  int age = 20;
  int* const pt = &age;

  int sage = 80;
  pt = &sage;  // ERROR pointer is constant
  *pt = 40;    // OK
  ```
Reference Type

- Reference?
  - An alternate name for an object
  - Actual implementation is the same as pointer in C
  - Frequently used for pass-by-reference

```c
void swap(int* x, int* y)
{
    int tmp = *x;
    *x = *y;
    *y = tmp;
}
```

Call-by-value in C
(pass pointers instead)

```cpp
void swap(int& x, int& y)
{
    int tmp = x;
    x = y;
    y = tmp;
}
```

Call-by-reference in C++
Reference Type

- A reference is an alias

```cpp
int a = 100;
int &rA = a; // initialized
rA += 1; // increase a to 101
```

- Reference variable **must be initialized**

```cpp
int &rB = 10; // Error
int &rC; // Error
```
Non-const reference cannot refer to const object

```
const int ten = 10;
const int &rTen = ten;
const int &r100 = 100;  // OK
int &rV = ten;         // Error
```

Any errors?

```
#include <iostream>
using namespace std;

int inc(int a) {
    const int &rA = a;
    rA++;
    return rA;
}
```
Class in C++

- **C++ class**
  - Similar to a `struct` in C (a list of fields/variables)
  - Additionally contains functions for the class

- **Object vs. Class**
  - Object is an `instance` of a class
  - Class is a `type` for a variable

```cpp
class Point { // declaration of class Point
    int x, y;

public:
    int add(int, int);
};

Point p, q; // two Point type objects
```
Headers

- Contains type declarations (.h)
  - Commonly included in multiple .cpp and .h files
  - Avoid multiple inclusion of headers:
    - Use `#include guards` or `#pragma once` (non-standard but ...)

```cpp
/* point.h */

#ifndef __POINT_H__
#define __POINT_H__

class Point {
    int x, y;

public:
    void set(const int, const int);
    int add(void);
};
#endif // __POINT_H__
```
Implementation Code

- Contains implementation of functions (.cpp)

```cpp
/* point.cpp */
#include "point.h"

void Point::set(const int a, const int b) {
    x = a;   y = b;
}

int Point::add(void) {
    return x + y;
}

/* main.cpp */
#include <iostream>
#include "point.h"
using namespace std;

int main(void) {
    Point p;
    p.set(1, 2);
    cout << "P: " << p.add() << endl;
    return 0;
}
```
Input Functions (more)

- getline (char* str, int strSize, char delim = ‘\n’);
  - Default delimiter: newline (‘\n’)

```
#include <iostream>
char line[80];
while (cin.getline(line, 80)) { ... }  // read every line
```

- stringstream class
  - Subclass of iostream  (www.cplusplus.com/reference)

```
#include <sstream>
stringstream ss;
ss.str(line);  // set stringstream from c-string
char token[80];
while (ss.getline(token, 80, ‘;’) { ... }  // read delimited token stream
```
Token String Stream

- Tokenized string stream example

```cpp
/* main.cpp */
#include <iostream>
#include <sstream>
using namespace std;

int main(void) {
    const int MAXLEN = 80;
    char line[MAXLEN], token[MAXLEN];
    int i = 0;

    while (cin.getline(line, MAXLEN)) {
        stringstream ss;
        ss.str(line);
        while (ss.getline(token, MAXLEN, ';')) {
            cout << i++ << ":" << token << endl;
        }
    }
}
```

<input>
apple; 100; 5;
fish; 200; 15;
milk; 50; 20;

<g++ main.cpp & run with input>
$ ./a.out < input
0:apple
1:100
2:5
3:fish
4:200;
5:15
6:milk
7:50
8:20
$