

# C++ programming

## Object oriented programming



Chris Trathen

**chrishuizhou-cplus@yahoo.com**

Lecture 1 24/02/2014

# If you cannot understand my English

- Read the lectures later. Translate to Chinese.
- Focus on the code examples.
- Get together with friends. Read my lectures.
- Find a good website on C++ written in Chinese.
- You are expected to be adults and think for yourself. So, find a way to understand everything.

# What I expect

- Attendance of all classes. Be on time. **Miss classes and you will lose marks.**
- Give 100% of your attention in class.
- Mobile phones may only be used if needed as English/Chinese dictionary. No phone calls or text messages.
- Ask questions.
- Give me feedback. What am I doing right and wrong?
- Do all practical work. **Finish at home.**

# No phone zombies (僵尸) allowed



# Two approaches

- Copy other people's work
  - Maybe get a Bachelor Degree in computer science.
  - Get a job making coffee at Starbucks.
- Think and try to do your own work
  - Get a Bachelor Degree in computer science.
  - Get a job in the IT industry and drink coffee at Starbucks.

# What you do each week

- 2 periods of lecture
- 2 periods of practical work
- 5 hours working at home
  - Several hours writing C++ programs.
  - Read my lectures and text book.
  - Work in small groups. Help each other.

# Improve yourself

- After you have written 1,000,000 lines of code, you are an expert in that language.
- I am an expert in some languages. I have written over 1,000,000 lines of code in Pick/basic and Smalltalk.
- I am not an expert in C++.
- I will NOT hand out solutions before you think for yourself.
- You will learn C++ by doing the exercises and by thinking.
- Find good C++ websites – read them. Use the code.
- Only use someone else's code if you understand it.
- Write lots of code – you will become a good programmer.
- **You only learn programming by doing it.**

# Subject aims

- Programming (object oriented) concepts.
- Programming paradigms: structured and object oriented.
- Foundation in C++ programming.
- Learning to think.



# Resources

- A copy of all lectures and other materials will be distributed by USB key or email.
- No paper handouts.
- Student monitors responsible for distribution of materials.
- Text book:
  - *The C++ programming language (4<sup>th</sup> ed)* by Bjarne Stroustrup. **Get PDF from me.**

# Practical class

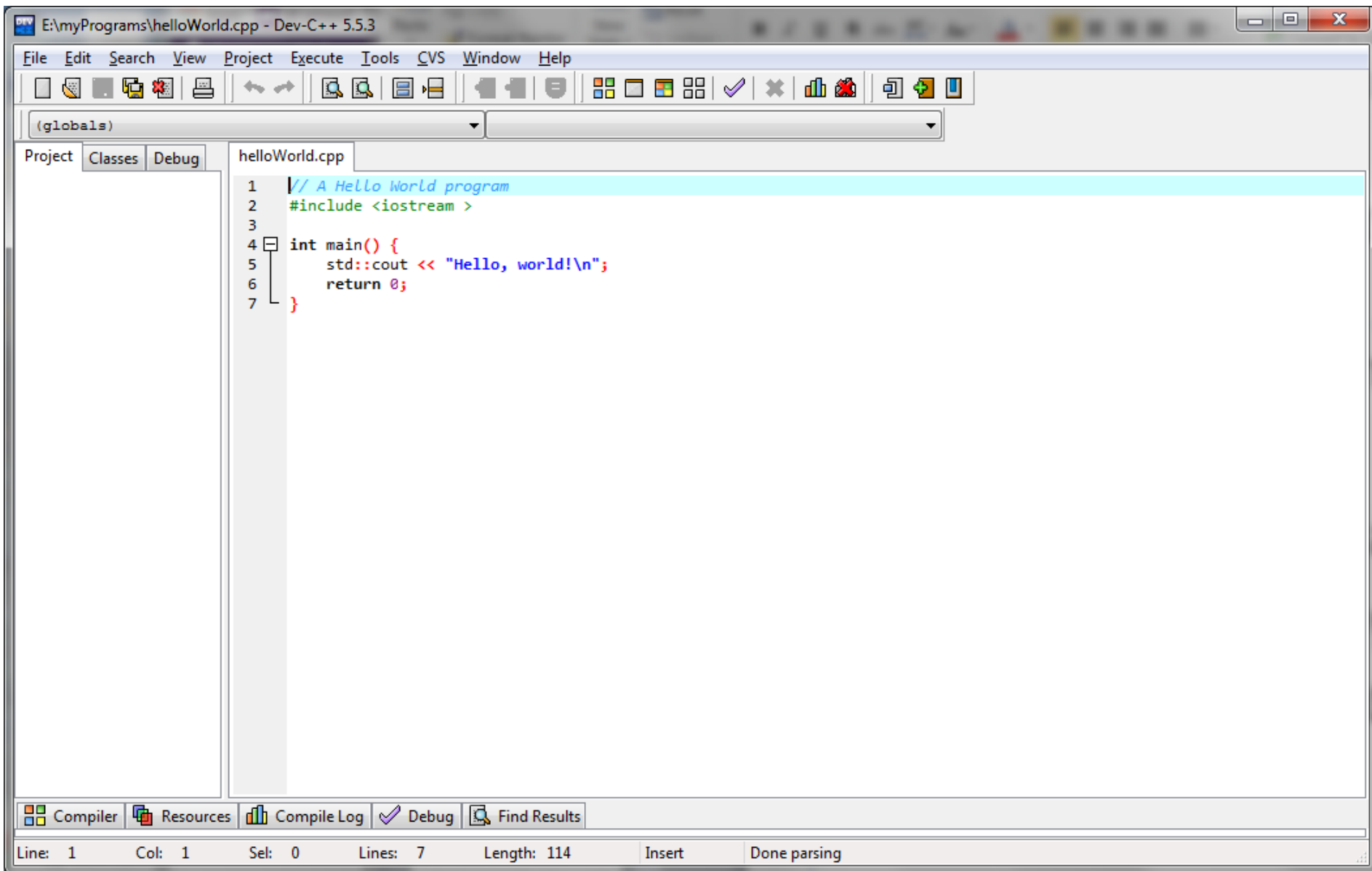
- The best way to learn programming is to do it.
- **Bring USB memory stick to all classes.**
- Bring C++ reference book on USB.
- Bring laptop if you want.
- Back up your work. Keep copies of everything.
- Exercises will be emailed or handed out via USB key.

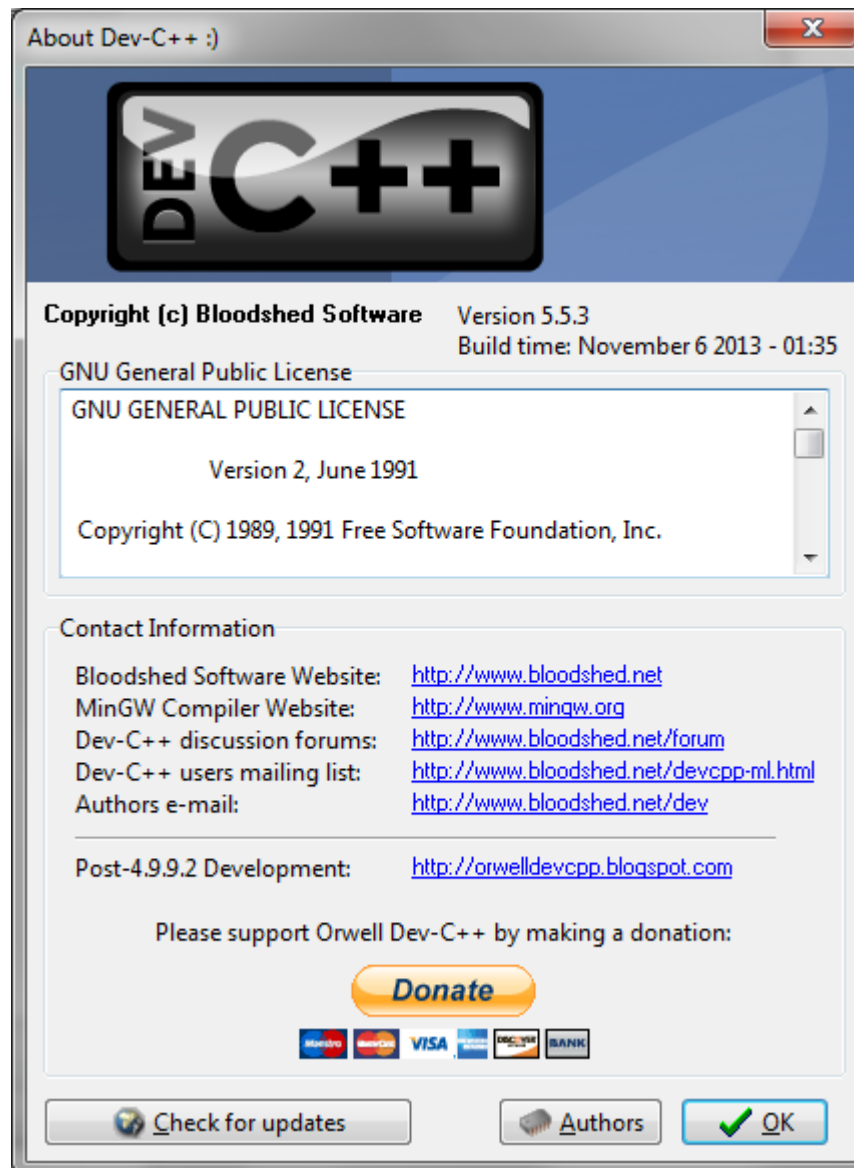
# Practical class

- Any C++ compiler can be used.
- **Orwell Dev C++** is free and can run on memory stick.

<http://sourceforge.net/projects/orwelldvcpp>

- Install file provided by me in the lab.





# Assessment

- Attendance of all classes, doing all homework and working hard.
- Completion of C++ project.
- Examination.
- More details later.

# To learn programming, listen to Albert Einstein:

- “Never memorize something that you can look up.”
- “Intelligence is not the ability to store information, but to know where to find it.”
- “A person who never made a mistake never tried anything new.”
- “Things should be as simple as possible, but no simpler.”

# Facts about software developers

- Very good developers are 20% more expensive, but 2000% more productive.
- 30-50% of experienced programmers cannot write programs.
- Good companies do not hire a programmer without a programming test.
- 1 woman needs 9 months to deliver a baby. 9 women do NOT deliver a baby in 1 month.
- Some of the best systems are written by just one or two programmers.



# Overview of programming languages

- Compiled languages
  - Translated to the target machine's native language by a compiler.
  - Faster execution.
  - Must be compiled for target machine (computer).
- Interpreted languages
  - Read by an interpreter and are executed by that program.
  - Slower execution.
  - Portable. Can be run on different machines.
- C++ is a compiled language.
- C++ compiles directly to a machine's native code, allowing it to be one of the fastest languages in the world.

# Type system

- A **type system** refers to the rules that the different types of variables of a language have to follow.
- A language can be typed or un-typed.
- Type can be checked at compile time or run time.
- C++ is a strongly typed language.

```
int x;
```

x is of type **int**

# Paradigms (范例)

- Declarative
  - Define the problem, not the solution.
- Functional
  - Express problems in terms of mathematical equations and functions.
- Structured
  - Use a structure based on sequence, selection, repetition.
- Object-oriented
  - Has classes which define the behaviour of their objects.
- ... other paradigms exist.
- C++ can support a few paradigms, particularly structured and object-oriented.

# Structured programming

- Contains 3 constructs:
  - Sequence
  - Selection
  - Repetition
- GOTO statement not allowed
- C++ has all three constructs. You can do structured programming in C++.

# Object Oriented programming

- Contains these concepts:
  - **Classification.** Objects belonging to classes.
  - **Inheritance.** Classes have sub-classes that inherit behaviour.
  - **Encapsulation.** Only the object may change itself.
  - **Polymorphism.** The same message can be sent to objects belonging to different classes.
- C++ has these concepts. You can do object oriented programming in C++. C++ is not a pure object oriented language.
- If you want to learn object oriented programming fully, teach yourself **Smalltalk**, a pure object oriented language.

# Why C++ ?

- C++ is an extension to C.
- C and C++ are suitable for writing operating systems, drivers, compilers, etc.
- C++ is **NOT** a good language for solving business problems. **Why?**

# C++

- C++ is an extension to C.
- You should know the basics of C from last semester.
- However, we will revise important basic ideas.

Bjarne Stroustrup, created C++ in the 1980s.





```
1 // Hello world program
2 #include <iostream>
3
4 int main() {
5     std::cout << "Hello World" << std::endl;
6     return 0;
7 }
```

1. This is a comment. The compiler ignores everything after `//` which can be at the start of a line or on the end of a valid statement.
2. The `#include` is called a compiler directive and tells the compiler to include a standard C++ library called `iostream`. Standard C++ libraries are always inside angle brackets `< >`.
3. You can use any amount of blank lines to make your program easier to read.
4. All C++ programs have one function called `main` and that is where the program starts running from. Everything from this line to line seven is part of `main`.
5. The `cout` operator is called to print out the text string "Hello World" followed by an `endl` (short for end line). The `<<` are used to separate items. The semi-colon identifies the end of the statement.
6. This line provides the return value from the `main` function.
7. This is the closing brace of the `main` function (and also the program).

```
// Hello world program
#include <iostream>

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

```
// Hello world program
#include <iostream>
using namespace std;
int main() {
    cout << "Hello world";
    return 0;
}
```

← Add std namespace. Simplify.

A *variable* represents a storage location. A variable has the following *attributes*:

- **Name** The *name* of a variable is the label used to identify a variable in the text of a program.
- **Address** The *address* of a variable is the memory address of the storage location(s) occupied by that variable.
- **Size** The *size* of a variable is the amount of storage (in bytes) occupied by that variable.
- **Type** The *type* of a variable determines the set of values that the variable can have *and* the set of operations that can be performed on that variable.
- **Value** The *value* of a variable is the content of the memory location(s) occupied by that variable. How the contents of the memory locations are interpreted is determined by the *type* of the variable).
- **Lifetime** The *lifetime* of a variable is the interval of time in the execution of a program during which a variable is said to exist. Some variables exist for the entire execution of a program; some are created and destroyed automatically during the execution of a program; and others are explicitly created and destroyed by the programmer.
- **Scope** The *scope* of a variable is the range of statements in the text of a program in which that variable can be referenced.

Consider the C++ variable declaration statement:

```
int i = 57;
```

- The **name** of the variable is i.
- The **type** of the variable is int,
- Its **size** is typically two or four bytes.
- Its initial **value** is 57.
- Some attributes of a variable, such its name, type and size, are bound at compile time. This is called *static binding*. Other attributes of a variable, such as its address and value, may be bound at run time. This is called *dynamic binding*.

# Type

- A variable can refer to simple values like integers called a **primitive type** or to a set of values called a **composite type**.
- A **composite type** is made up of **primitive types** and other **composite types**.
- A variable must declare what type it is before it can be used in order to enforce value and operation safety and to know how much space is needed to store a value.

# Primitive types in C++

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Several of the basic types can be modified using one or more of these type modifiers

- signed
- unsigned
- short
- long

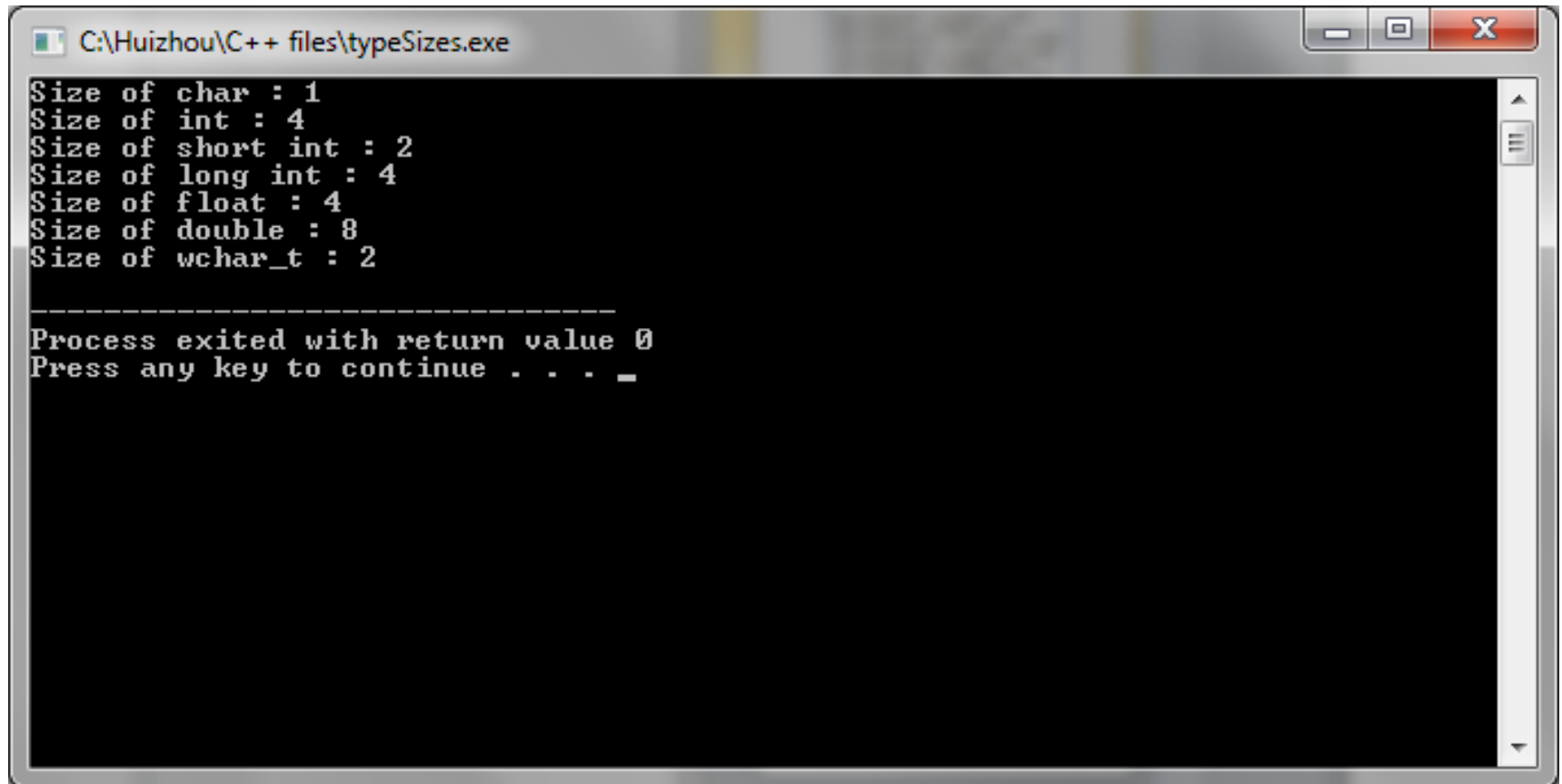
Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character



# Program to check type size in bytes

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

int main()
{
    cout << "Size of char : " << sizeof(char) << endl;
    cout << "Size of int : " << sizeof(int) << endl;
    cout << "Size of short int : " << sizeof(short int) << endl;
    cout << "Size of long int : " << sizeof(long int) << endl;
    cout << "Size of float : " << sizeof(float) << endl;
    cout << "Size of double : " << sizeof(double) << endl;
    cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;
    return 0;
}
```



```
C:\Huizhou\C++ files\typeSizes.exe  
Size of char : 1  
Size of int : 4  
Size of short int : 2  
Size of long int : 4  
Size of float : 4  
Size of double : 8  
Size of wchar_t : 2  
  
-----  
Process exited with return value 0  
Press any key to continue . . . _
```

# Variable Definition in C++

```
int i, j, k;      //declaration of variables  
char c, ch;  
float f, salary;  
double d;
```

```
int d = 3, f = 5; // definition and initializing values together  
byte z = 22;  
char x = 'x';  
bool a = true;
```

# C++ Variable Scope

- A scope is a region of the program. There are three places, where variables can be declared:
  - Inside a function or a block which is called local variables.
  - In the definition of function parameters which is called formal parameters.
  - Outside of all functions which is called global variables.

# C++ Variable Scope – local variables

- Variable definitions are considered to be executable statements
- When a local variable is defined - it is not initialised by the system, you must initialise it yourself.
- A local variable is defined inside a block and is only visible from within the block.

```

main()
{
    int i=4;
    int j=10;

    i++;

    if (j > 0)
    {
        printf("i is %d\n",i);    // i (value 5) defined in 'main' can be seen
    }

    if (j > 0)
    {
        int i=100;                // 'i' is defined and so local to this block
        printf("i is %d\n",i);    // i (value 100) can be seen
    }                            // 'i' (value 100) dies here

    printf("i is %d\n",i);        // 'i' (value 5) is now visible.
}

```

# C++ Variable Scope – global variables

- Global variables are initialised by the system when you define them.

<b>Data Type</b>	<b>Initialiser</b>
int	0
char	'\0'
float	0
pointer	NULL

```
int i=4;      // Global definition. i is global variable.
```

```
main()
{
    i++;      // global variable i is accessed.
    func;
}
```

```
func()
{
    int i=10; // local declaration. This i is local.
    i++;      // local variable i.
}
```

← Local variable i has a scope of func() only.



# Homework

- You tell me.