C++ Language Rules

PHYS2G03

McMaster University

C++ Language Overview

C++ is based on C

C++ invented by Bjarne Stroustrup

"The C++ Programming Language" (1986)

Designed to facilitate object oriented programming

Computer Scientist – Worked on Intelligent Agents

First C++ Standard appeared in 1997 (!)

99.9% of C is legal in C++ -- in general only obscure parts of C will not work

C was invented to code Unix (~1970). First standardized in 1989!

Standard text: Kernighan & Ritchie 1988

"The C Programming Language"

All language standards evolve over time. We will stick to the most standard features of C/C++

C++ Language Overview

Each language has formal rules for what constitutes legal code

The compiler tries to understand your programs based on these rules. If it can't you get **syntax errors** at compile time

C/C++ Terms

Keywords: (also called **reserved words**)

Part of the language, can't be used for a variable or function name

alignas (since C++11)	enum	return
alignof (since C++11)	explicit	short
and	export(1)	signed
and_eq	extern	sizeof
asm	false	static
auto(1)	float	static_assert (since C++11)
bitand	for	static_cast
bitor	friend	struct
bool	goto	switch
break	if	template
case	inline	this
catch	int	thread_local (since C++11)
char	long	throw
char16_t (since C++11)	mutable	true
char32_t (since C++11)	namespace	try
class	new	typedef
compl	noexcept (since C++11)	typeid
const	not	typename
constexpr (since C++11)	not_eq	union
const_cast	nullptr (since C++11)	unsigned
continue	operator	using(1)
decltype (since C++11)	or	virtual
default(1)	or_eq	void
delete(1)	private	volatile
do	protected	wchar_t
double	public	while
dynamic_cast	register	xor
else	reinterpret_cast	xor_eq

calc.cpp

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

calc.cpp is an example program

You can get a copy from /home/2G03/calc

calc.cpp Keywords

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

calc.cpp Keywords

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them a
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integ
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

#include is not a C++ keyword
It is reserved for the compiler though
Lines with # are compiler directives used in several languages.
In this case it inserts <iostream> which has declarations for code associated with input and output such as std::cout

C/C++ Basic Rules

- C/C++ is case sensitive
- C/C++ doesn't care about "whitespace"

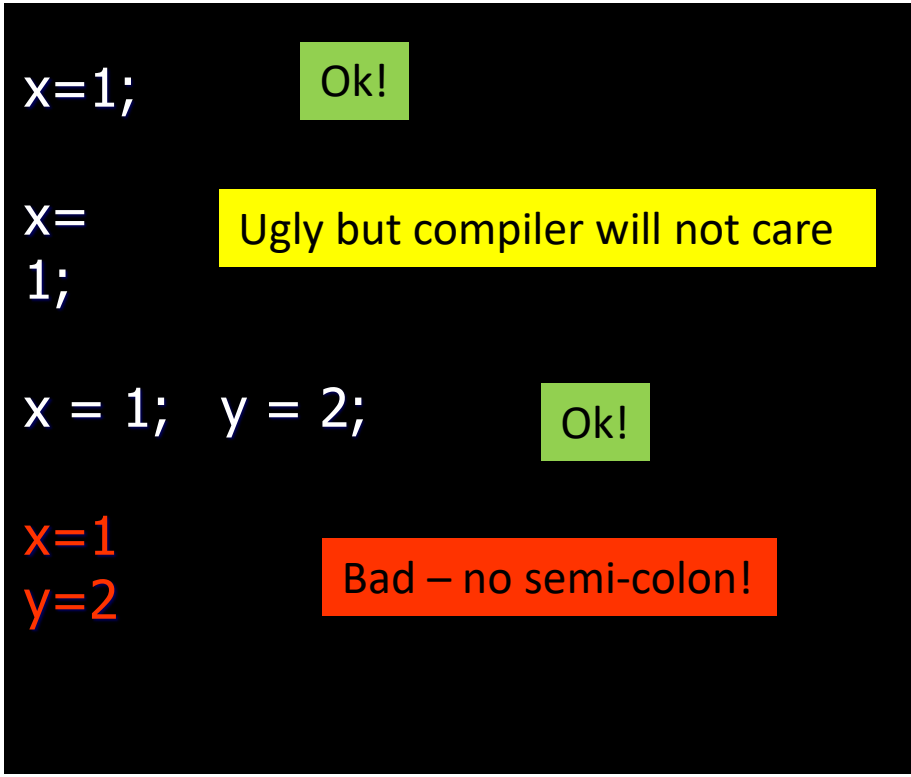
There can be any number of spaces, tabs or lines before or between keywords, operators (+) and variable names (a, x, …)

Note: You cannot put spaces in a name:

int apple; is not the same as int a pple;

C/C++ Basic Rules

- C/C++ expects every statement to end with ;
  ; is the semi-colon
- Statement are actual code to do something
- This is how C/C++ decides something is done – not spaces or newlines



C/C++ Basic Rules

- Any text following two slashes is a comment

The compiler ignores text after the // to the end of the line

- Old style comments look like this

  /* Comment */

Modern C and C++ compilers accept either

```
// This is a comment that is just one line
// So it this

/*  This is a comment that keeps on until
    it sees a star followed by a slash */

   a = 2;
   b = 3;   // This is a trailing comment

// c = 2;    Code ignored as a comment
```

calc.cpp

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

Compiler directive – not actually C++

Function declaration, not a code statement

Comments – no ;

statement ; required

C/C++ Terms

Variable: A word naming a variable

e.g. x, i, day, half_life, r2, f0

- Must start with a letter or underscore _ can include _ letters and numbers

Note! In C/C++ case matters

Half_Life is different to Half_life and half_life

Choose a consistent naming scheme so it is easy to remember

C/C++ Terms

Type: C/C++ variables are **statically typed**
Variables must have declared types

Standard types:	integers	int, long, short
	real	float, double
	logical	bool
	text	char, string

The different kinds of integers and real numbers refer to how much memory should be devoted to storing the variable. More memory means more precision. (more later…)

You can define your own types. Objects are special user defined types. Users provide the names for functions, variable and new types. Choose sensible names to avoid confusion..

C/C++ static types

Static types makes life a lot easier for the compiler and often correspond directly to what hardware can do. The code is generally much faster.

e.g.

```
int a;     // a is an integer and can only store integers.
a=1;   // ok
a=2.5;  // ok, but converted to integer, 2 is stored
a="frog"; // error
```

Scripting languages are often **dynamically** typed (e.g. Python, shell languages like tcsh):

```
x=2          # x is an integer now!
x="frog"   # x is a string now!
```

calc.cpp Types & Variables

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

calc.cpp Types & Variables

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums th
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

main is a name of a function not a variable. It is similar to a variable in that it is a named thing (names should be unique)

std::cout is a variable defined in iostream

(std::cout is actually an object which is a special kind of variable defined in iostream. We will later explore the extra aspects that make it an object)

C/C++ Terms

Constant: A text string or numeric constant (integer or real number)

You can't set a value for it:

```
a = 2;  //ok
2 = a;  //not ok
```

You can specially name constants if you like:

```
const float k_B=1.38066e-16;
k_B=3;   //not ok now
```

C/C++ Terms

Constant: A text string or numeric constant (e.g. integer or real number)

No decimal point means integer, e.g. 2

Decimal or exponent means real number
e.g. 2.0, 2e0 or 2.
1.333e-10 is 1.333×10^{-10}
(Stored as a double by default)

"Hello World" a string constant
true a bool constant

calc.cpp Constants

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

0 is an integer constant

text in quotes is a string constant

C/C++ Terms

Operator: Part of the language, may indicate a math operation, assignment, comparison …

- Math: plus +, multiply *, divide /
- Assignment: =
- Comparison: == Equal, > Greater than, >= Greater than or equal to

In C++ (and other object oriented langues like Fortran 90) you can invent new operators: e.g. << in iostream

C/C++ Terms

Operator: Operators operate on operands – the variables or constants on either side

Not every combination is legitimate

e.g. “a” + 1 doesn’t work

1 + x = 2 illegal

Operators generally produce a new value that is *usually* of the same type

1+1 gives 2 (int + int gives int)

1+1.0 gives 2.0 (int + double gives double)

calc.cpp Operators

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

+ is a math operator
It will add together integers or real numbers depending on context
= is the assignment operator
Not just for math!

calc.cpp Operators

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

<< and >> are made-up operators (no math equivalent)
Stroustrup invented them as a way to print with the std::cout object (and do input with std::cin object)

C/C++ Terms

Expression: A collection of variables or constants connected with operators

e.g. a+b

(a+b+2)

1.0 + sin(x)

(a < b) logical expression

Expressions are typically stored in a variable or tested for being true or false

e.g. x = 1.0+sin(x);

```
if (a<b) {  // do stuff
}
```

C/C++ Terms

A **block** is code in between braces

```
{
    std::cout << “Hello World!\n”;
}
```

```
{  x=1; y=2; }
```

Remember: C/C++ doesn’t care about spaces or new lines, only ; and { }

calc.cpp Code blocks

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  // 2. Sum a and b
  c = a + b;

  // 3. Output the results
  std::cout << "The sum of " << a << " and " << b << " equals " << c << "\n";
  return 0; // success
}
```

The { } indicate the block of code belonging to the main function

C/C++ Terms

blocks are a way to associate several bits of code with one construct (e.g. a function) using braces {}

e.g.

main() { *all the code in the main function* }

LOOP (repeating code):

while (true) { *all the code that is repeated* }

BRANCH (conditional code):

if (r==0) { *all the code that is used if r==0 is true* }

branch Code blocks

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  if (a == b) {
      std::cout << “Please provide two different integers\b”;
      return 1;
   }
   return 0;
}
```

Note the nested block inside the main block

Loops

LOOP (repeating code):

while (true) {

all the code that is repeated

}

for (;;) { *all the code that is repeated* }

C/C++ uses **while** and **for** statements to do loops. The loops above go forever. In general there would be conditions inside the brackets saying when to stop.

break is another way to get out of a loop.

loop Code blocks

```
#include <iostream>
int main()
{
  // Calc program
  // Takes two integers, sums them and reports the answer
  int a,b,c;

  // 1. Input 2 integers
  std::cout << "Please input two integers\n";
  std::cin >> a >> b;

  while (a<b) {
      a=a+1;
      std::cout << “now a=” << a << “ b=“ << b << “\n”;
   }
   return 0;
}
```