Syntax and Semantics

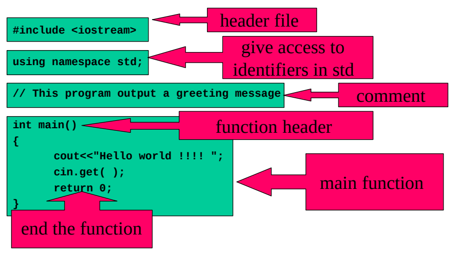
Outline

- C++ Structure
- Data types

Terminology

- A *programming language* is a set of rules, symbols, and special words used to construct a program.
- *Syntax* (grammar) is a set of rules that precisely states how valid instructions to be constructed in C++.
- *Semantics* (meaning) is the correctness of instructions written in C++.

C++ Structure



```
#include <iostream>

using namespace std;

// This program output a greeting message

int main()
{
      cout<<"Hello world !!!! ";
      cin.get( );
      return 0;
}
```

C++ Program

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

/* This program computes the sum and product of two integers. */

int main(void)
{
  int Number1, Number2, Sum, Product;
  Number1 = -5;
  Number2 = 10;
  Sum = Number1 + Number2;
  Product = Number1 * Number2;
  cout<<"The sum is "<<Sum<<'\n';
  cout<<"The product is "<<Product;
  return 0;
}
```

C++ Structure

- All C++ programs have
  - header file(s)
  - a *function* called *main.*
- *Function* is a subprogram in C++.
- *Begin* { *and end* } *markers*: to indicate the beginning and ending of a block of statements to be executed.

Identifiers

An *identifier* is a name associated with a function or data object (variable, data type).

- Combinations of letters (A…Z, a…z), digits (0…9), and underscore (_)
- Must begin with a letter or underscore
- No special characters such as +, $, , *, ', etc.
- Case sensitive

Examples

- Valid identifier:
  - Value2, Sum, Integer1, Product, Total_Income
- Invalid Identifier:
  - Number 1, 2Data, First-Initial, Cost_in_$, float

Reserved Words

A *reserved word* is a predefined word with a special meaning in C++ (appendix A-1), e.g., int, if, else, for, switch...

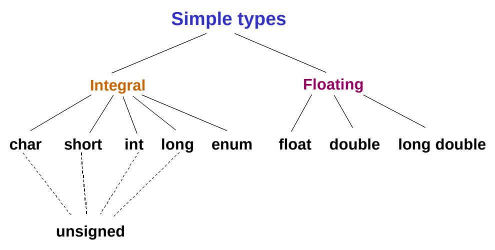
Data and Data Types

- Each data in C++ has a type associated with it.
- A *data type* is a set of data together with a set of operations on the data values, e.g., char, short, int, float, long...

Classifications of Data types

- *Integral type*:
  - An *integral type* is a data type possesses integer values, e.g., char, short, int, and long.
- *floating type*:
  - A *floating type* is a data type with a set of operations on real numbers, e.g., float, double (double precision), and long double.

C++ Simple Data Types



Integral Types

- int
  - 123, +123, -123, 22333
- char
  - ‘S’, ‘c’, ‘4’, ‘@’

Integral Types

Type	Size in Bytes	Minimum Value	Maximum
char	1	-128	127
unsigned char	1	0U	255U
short	2	-32,768	32,767
unsigned short	2	0U	65,535U
int	4	-2147483648	2147483647
unsigned int	4	0U	4,294,967,295U
long	8	-9223372036854775808L	9223372036854775807L
unsigned long	8	0UL	18446744073709551616UL

Floating-Point Types

A real number or a floating number has an integer part and a fractional part, with a decimal point in between e.g., 18.0, 127.54, 0.57, 4., .8, 1.74536E-12, 3.652442E4, 7E20.

Type	Size in Bytes	Minimum Value	Maximum
float	4	3.4E-38	3.4E+38
double	8	1.7E-308	1.7E+308
long double	10	3.4E-4932	1.1E+4932

Variables

- A *variable* is an identifier associated with a memory location that used to store data, e.g.,

Data1

(memory location: 1110101101)

65

Declaration

- A statement that associates an identifier with a data object, a function, or a data type is called *declaration statement.*

data type Identifier, Identifier, …;

- `int Number1;`
- `int main();`
- `char FirstInitial;`
- `float Wages;`

Constants

A *constant variable* is an identifier with a fixed value.

const data type identifier = literal value

- `const float PI = 3.14159;`
- `const int MaxHours = 40;`
- `const char Blank = ' ';`

Assignment Statement

- An *assignment statement* is a statement that assigns the value of an expression into a variable.

Variable = Expression;

Expression: an arithmetic expression, another identifier or a literal value

Examples

```
– Count = 153;
– Count = Count1;
– Num1 = Num2*Num3/Num4;
– MidInit = 'A';
```

More Examples

```
{
   ...
   int Number1;
   char FirstInitial;
   const float HourRate = 5.65;
   float Wages;
   int Hours;

   Number1 = 65;
   FirstInitial = 'a';
   Wages = HourRate*Hours;
   ...
}
```

C++ Preprocessor

Numerous libraries of functions are included as part of any C++ programming package. The actual code for these libraries has been already compiled and to be added to the program during the compiling phase of program construction. We use *compiler directive* **`#include`** to inform the compiler which libraries are needed or to be included in the source program.

Examples

#include <iostream>
#include <iomanip>
#include <iomanip>
#include <cmath>
#include <string>

iostream is called a *header file* whose contents are inserted in place of the **`#include`** line during compilation. These header files can be found in *c:\Program Files\Microsoft Visual Studio\Vc98\Include*

Namespace

- A **`namespace`** contains identifiers that used or declared in header files.
- The namespace for identifiers in these header files is called *std*
- The purpose of a **`namespace`** is to provide a mechanism that minimizes the possibility of accidentally duplicating names in various parts of a program.

Namespace directive

`using namespace std;`

`using` directive in the line indicates that there are identifiers in the **`namespace`** called **`std`** will be used in the program.

Another way to reference identifiers in the namespace **`std`** is by using a *qualified* name

`std :: cout`

Mathematical Operations

+	**Unary plus**
-	**Unary minus**
+	**Addition**
-	**Subtraction**
*	**Multiplication**
/	**Division**
%	**Modulus**

A *unary operator* is an operator with one operand.
A *binary operator* is an operator with two operands.

Precedence of C++ Operators

- / % *
- + –

Examples

```
Sum = Num1+ Num2;
Average = (Num1+ Num2)/2;
W = X * Y / X + Y;
Delta = B * B – 4 * A * C;
Quotient = X / Y;
Remainder = X % Y;
```

Type of Arithmetic Expressions

combining float with either integer or float

⇒ float

combining integers or floats with /

⇒ integer or float

combining integers with either +, –, *

⇒ integer

% can be used with only integer

Modulus %

The modulus operator % yields the *integer remainder* of the result of dividing its first operand by its second. If either one of the operands is negative, the sign of the result is machine-dependent.

Examples

14 % 3   = 2
-14 % 3   = 2 or -2
14 % -3   = 2 or -2
-14 % -3 = 2 or -2

Increment and Decrement Operators

+ + Increment
– – decrement
Examples:
Num + + or + + Num ⇔ Num = Num + 1