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

```cpp
#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<<'
';
    cout<<"The product is "<<Product;
    return 0;
}
```

Another C++ Program

```cpp
#include <iostream>

int Square( int ); // declares these two
int Cube( int ); // value-returning functions
using namespace std;

int main( )
{   // function call
    cout << "The square of 27 is " << Square(27) << endl;
    // function call
    cout << "The cube of 27 is " << Cube(27) << endl;
    return 0;
}

int Square( int n )
{   return n * n;
}

int Cube( int n )
{   return n * n * n;
}
```

Output of program

The square of 27 is 729
The cube of 27 is 19683

Program With Several Functions

```
main function
square function
cube function
```
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.

C++ function Structure

```
int main ( )
{
    return 0;
}
```

What is in a heading?

- **Type of returned value**: `int`
- **Name of function**: `main`
- **Says no parameters**

Block (Compound Statement)

- **A block** is a sequence of zero or more statements enclosed by a pair of curly braces `{ }`

SYNTAX

```c
{
    Statement (optional)
    
    
}
```
C++ function Structure

Return Type

```
int main ( )
{
    Function Name
    Statements;
    •
    •
    return 0;  // Exit statement
}
```

Exit status

- The integer value returned by `main` to the operating system when it completes execution
- Exit status of 0 indicates successful completion of the program on most computer systems
- Nonzero value indicates an error

Function Concept in Math

```
f ( x ) = 5 x - 3
```

When $x = 1$, $f ( x ) = 2$ is the returned value. When $x = 4$, $f ( x ) = 17$ is the returned value. Returned value is determined by the function definition and by the values of any parameters.

Functions

In C++ there are two types of functions, *predefined functions* and *user-defined functions*.
- Predefined or built-in functions come with libraries of C++.
- User-defined functions are created by programmers.
Predefined Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Type of Arguments</th>
<th>Type of Value Returned</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqrt(x)</td>
<td>double</td>
<td>double</td>
<td>square root of x</td>
</tr>
<tr>
<td>floor(x)</td>
<td>double</td>
<td>double</td>
<td>largest integer &lt; x</td>
</tr>
<tr>
<td>ceil(x)</td>
<td>double</td>
<td>double</td>
<td>smallest integer &gt; x</td>
</tr>
</tbody>
</table>

Total Weight = floor(Weight);
Total Amount = ceil(SubTotal1 + SubTotal2);

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

More About Identifiers

- Some C++ compilers recognize only the first 32 characters of an identifier as significant
- Then these identifiers are considered the same:
  
age_Of_This_Old_Rhinoceros_At_My_Zoo
  age_Of_This_Old_Rhinoceros_At_My_Safari

- Consider these:
  
  Age_Of_This_Old_Rhinoceros_At_My_Zoo
  age_Of_This_Old_Rhinoceros_At_My_Zoo

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., \texttt{int}, \texttt{if}, \texttt{else}, \texttt{for}, \texttt{switch}...

Data and Data Types

\begin{itemize}
  \item Each data in C++ has a type associated with it.
  \item A data type is a set of data together with a set of operations on the data values, e.g., \texttt{char}, \texttt{short}, \texttt{int}, \texttt{float}, \texttt{long}...
\end{itemize}

Classifications of Data types

\begin{itemize}
  \item Integral type:
    \begin{itemize}
      \item An integral type is a data type possesses integer values, e.g., \texttt{char}, \texttt{short}, \texttt{int}, and \texttt{long}.
    \end{itemize}
  \item Floating type:
    \begin{itemize}
      \item A floating type is a data type with a set of operations on real numbers, e.g., \texttt{float}, \texttt{double} (double precision), and \texttt{long double}.
    \end{itemize}
\end{itemize}

C++ Data Types

\begin{itemize}
  \begin{itemize}
    \item Simple types: \texttt{char}, \texttt{short}, \texttt{int}, \texttt{long}, \texttt{bool}, \texttt{float}, \texttt{double}, \texttt{long double}
    \item Structured types: \texttt{array}, \texttt{struct}, \texttt{union}, \texttt{class}
    \item Pointer types: \texttt{char}, \texttt{short}, \texttt{int}, \texttt{long}, \texttt{bool}, \texttt{float}, \texttt{double}, \texttt{long double}
    \item Reference types
  \end{itemize}
\end{itemize}
C++ Simple Data Types

Simple types

Integral

char short int long enum float double long double

Floating

char short int long enum float double long double

Integral Types

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

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.

<table>
<thead>
<tr>
<th>Type</th>
<th>Size in Bytes</th>
<th>Minimum Value</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4</td>
<td>3.4E-38</td>
<td>3.4E+38</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>1.7E-308</td>
<td>1.7E+308</td>
</tr>
<tr>
<td>long double</td>
<td>10</td>
<td>3.4E-4932</td>
<td>1.1E+4932</td>
</tr>
</tbody>
</table>
Variables

- A **variable** is a location in memory that stores a **data value** which can be changed and reference by an identifier, e.g.,

  Data1
  (memory location: 125=1111101)

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

A declaration tells the compiler to **allocate** enough memory to hold a value of this data type and to **associate** the identifier with this location

```plaintext
data type Identifier, Identifier, ...
```

- int Number1; allocate 4 bytes for Number1
- char FirstInitial; allocate 1 byte for FirstInitial
- float Wages; allocate 4 bytes for Wages

Constants

- A **constant variable** a location in memory that we can refer to by an identifier, and in which a **data value that cannot be changed** is stored.

```plaintext
const data type identifier = literal value
```

- `const float PI = 3.14159;`
- `const int MaxHours = 40;`
- `const char Blank = ' ';`
- `const string STARS = "*****";`

C++ Data Type String

- A string is a sequence of characters enclosed in double quotes
- String sample values
  - "Hello"
  - "Year 2000"
  - "1234"
  - "Joe Nobody"
- The empty string (null string) contains no characters and is written as ""
Strings

- A value of type char is limited to a single character.
- A string is a sequence of characters enclosed in double quotes, e.g., "Problem Solving"
- "C++"
- "Programming and ."
- " . . . ."
- A string must be typed entirely on one line. The following string is invalid:
  "This string is invalid because it is typed on more than one line."

More About Type String

- String is not a built-in (standard) type
  - it is a programmer-defined data type
  - it is provided in the C++ standard library
- String operations include
  - comparing 2 string values
  - searching a string for a particular character
  - joining one string to another

What is an Expression in C++?

- An expression is a valid arrangement of variables, constants, and operators.
- In C++ each expression can be evaluated to compute a value of a given type.
- The value of the expression 9 + 5 is 14

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

<table>
<thead>
<tr>
<th>/</th>
<th>%</th>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

### 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
  - $\Rightarrow$ float
- Combining integers or floats with `/`
  - $\Rightarrow$ integer or float
- Combining integers with either `+`, `-`, `*`
  - $\Rightarrow$ integer
- `%` can be used with only integer

### Division Operator

- `11 / 4 = 2`
- `11.0 / 4.0 = 2.75`
- `11 / 4.0 = 2.75`
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
14 % -3 = 2
-14 % -3 = -2

Increment and Decrement

++ Increment
-- decrement

Examples:

Num ++ or ++ Num => Num = Num + 1

Assignment Statement

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

```
Variable = Expression;
```

- Expression is an arithmetic expression, another identifier or a literal value
- first, Expression on right is evaluated
- then the resulting value is stored in the memory location of Variable on left
- An automatic type coercion occurs after evaluation but before the value is stored if the types differ for Expression and Variable.
Example

```cpp
int age;
age = 8;
age = age + 1;
```

PREFIX FORM
Increment Operator

```cpp
int age;
age = 8;
++age;
```

POSTFIX FORM
Increment Operator

```cpp
int age;
age = 8;
age++; 
```

Decrement Operator

```cpp
int dogs;
dogs = 100;
dogs--; 
```
When the increment (or decrement) operator is used in a "stand alone" statement solely to add one (or subtract one) from a variable's value, it can be used in either prefix or postfix form.

```
// USE EITHER
--dogs;     dogs--;
```

When the increment (or decrement) operator is used in a statement with other operators, the prefix and postfix forms can yield different results.

```
WE'LL SEE HOW LATER . . .
```

What value is stored?

```
float a;
float b;

a = 8.5;
b = 9.37;
a = b;
```

What is stored?

```
float someFloat;

someFloat = 12; //causes implicit type conversion
```

```
float someFloat;

someFloat = 12; //causes implicit type conversion
```
What is stored?

```cpp
int someInt;
```

```cpp
? 3
```

```cpp
someInt = 4.8; //causes implicit type conversion
```

```cpp
4
```

type casting is explicit conversion

```cpp
int(4.8) = 4
float(5) = 5.0
float(7/4) = 1.0
float(7)/float(4) = 1.75
```

Some Expressions

```cpp
int age;
EXAMPLE VALUE
age = 8 8
-age =-8
5 + 8 13
5 / 8 0
6.0 / 5.0 1.2
float (4 / 8) 0.0
float (4) / 8 0.5
```

What values are stored?

```cpp
float loCost;
float hiCost;
loCost = 12.342;
hiCost = 12.348;
loCost = float(int(loCost*100.0+0.5))/100.0;
hiCost = float(int(hiCost*100.0+0.5))/100.0;
```

```cpp
12.34 12.35
```

```cpp
loCost hiCost
```
### Examples

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

### More Examples

```c++
[ ... int Number1;
 char FirstInitial;
 const float HourRate = 5.65;
 float Wages;
 int Hours;
 Number1  = 65;
 FirstInitial  = 'a';
 Wages = HourRate*Hours;
 ... ]
```

### String Concatenation (+)

- Concatenation is a binary operation that uses the + operator.
- At least one of the operands must be a string variable or named constant—the other operand can be string type or char type.

### Concatenation Example

```c++
const   string WHEN = "Tomorrow";
const   char EXCLAMATION = '!' ;
string  Message1 ;
string  Message2 ;
Message1 = "Yesterday " ;
Message2 = "and " ;
Message1 = Message1 + Message2 +
           WHEN + EXCLAMATION ;
```
Insertion Operator `<<`

- Variable `cout` is predefined to denote an output stream that goes to the standard output device (display screen).
- The insertion operator `<<` called “put to” takes 2 operands.
- The left operand is a stream expression, such as `cout`. The right operand is an expression of simple type or a string constant.

Output Statements

Syntax

```
cout << Expression << Expression << ...
```

These examples yield the same output:

```
cout << "The answer is ";
cout << 3 * 4 ;
cout << "The answer is " << 3 * 4;
```

No I/O is built into C++

Instead, a library provides an output stream.

Is compilation the first step?

- No. Before your source program is compiled, it is first examined by the preprocessor to:
  - Remove all comments from source code.
  - Handle all preprocessor directives—these begin with the `#` character such as `#include <iostream>`.
  - Tell preprocessor to look in the standard include directory for the header file called `iostream` and insert its contents into your source code.
Using Libraries

- A library has 2 parts
  - Interface (stored in a header file) tells what items are in the library and how to use them.
  - Implementation (stored in another file) contains the definitions of the items in the library.

- `#include <iostream>`
  - Refers to the header file for the `iostream` library needed for use of `cout` and `endl`.

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

```cpp
#include <iostream>
#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

The `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`

---

C++ Program

```
// ****************************************************
//  PrintName program
//  This program prints a name in two different formats
// ****************************************************

#include <iostream> // for cout and endl
#include <string> // for data type string
using namespace std;

const string FIRST = "Herman"; // Person's first name
const string LAST = "Smith"; // Person's last name
const char MIDDLE = 'G'; // Person's middle initial

int main()
{
    string firstLast; // Name in first-last format
    string lastFirst; // Name in last-first format
    firstLast = FIRST + " " + LAST;
    cout << "Name in first-last format is " << endl << firstLast << endl;
    lastFirst = LAST + " , " + FIRST + " ";
    cout << "Name in first-last format is " << endl << lastFirst << MIDDLE << ". " << endl;
    return 0;
}
```

---

Output of Program

```
Name in first-last format is
Herman Smith
Name in last-first-initial format is
Smith, Herman G.
```