C++ Part1 : Introduction, Variables and Functions

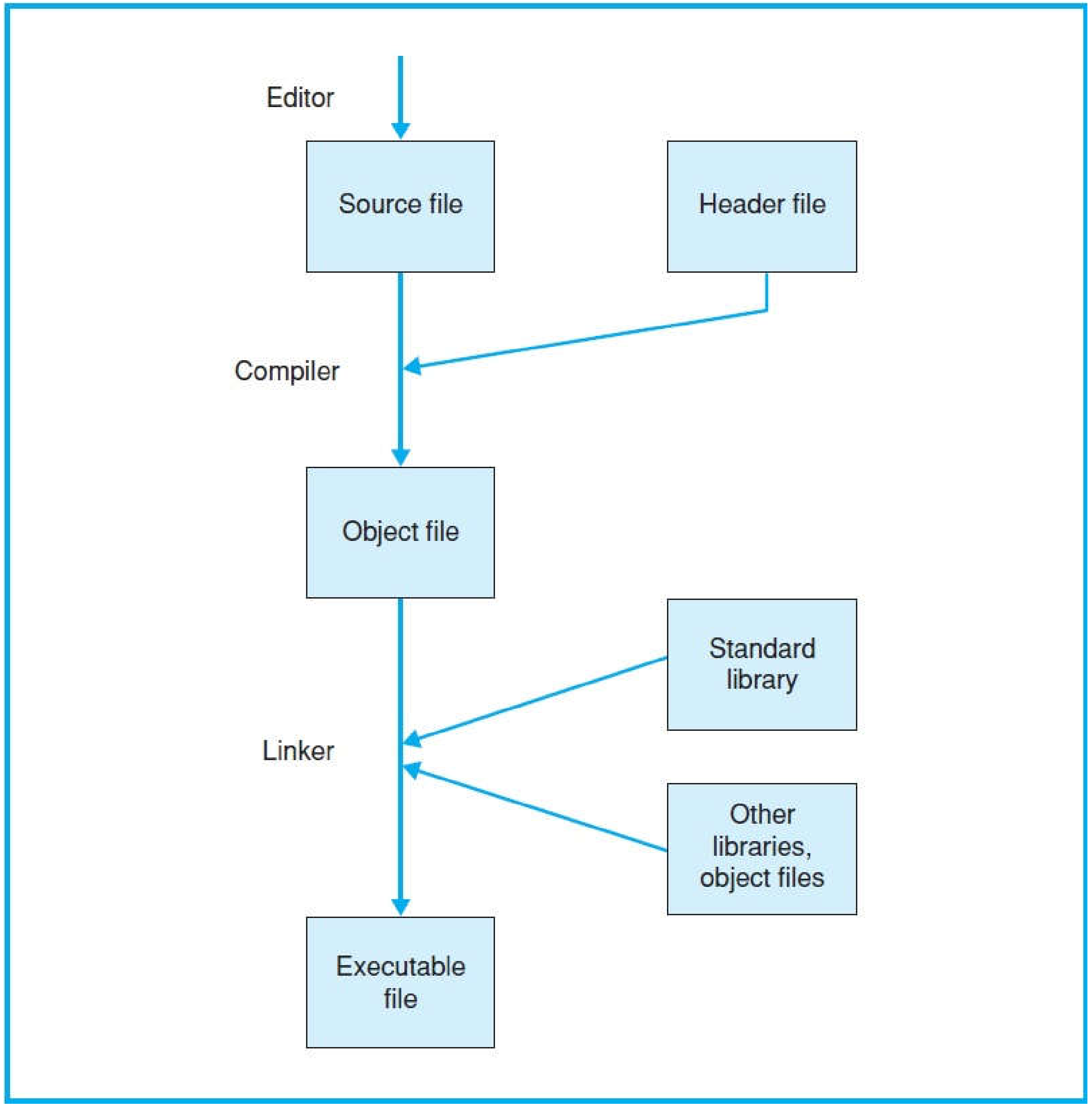
Parts of a Program

- Documentation Section
- Preprocessor Directives
  - Link Section
  - Definition Section
- Global Declaration Section
- Global Function: Main
- Program Body
- Statements
  - Local variable declaration statement
  - Assignment statement
  - Input/output statements
  - Return statement

```
// my first program in C++
#include<iostream>
#define NAME ”Livewire”
using namespace std;

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

C++ Program Execution



Fundamental Data types

- Boolean (bool)
- Character (char)
- Integer (int)
- Floating Point (float)
- Double Floating Point (Double)
- Wide Character (wchar_t)
- Valueless (void)

- Modifiers
  - Long
  - Short
  - Long Long (C++ 11)
  - Signed
  - Unsigned

Group	Type names*	Notes on size / precision
Character types	char	Exactly one byte in size. At least 8 bits.
	char16_t	Not smaller than char. At least 16 bits.
	char32_t	Not smaller than char16_t. At least 32 bits.
	wchar_t	Can represent the largest supported character set.
Integer types (signed)	signed char	Same size as char. At least 8 bits.
	signed short int	Not smaller than char. At least 16 bits.
	signed int	Not smaller than short. At least 16 bits.
	signed long int	Not smaller than int. At least 32 bits.
	signed long long int	Not smaller than long. At least 64 bits.
Integer types (unsigned)	unsigned char	(same size as their signed counterparts)
	unsigned short int	
	unsigned int	
	unsigned long int	
	unsigned long long int	
Floating-point types	float	
	double	Precision not less than float
	long double	Precision not less than double
Boolean type	bool	
Void type	void	no storage
Null pointer (since c++11)	decltype(nullptr)	
Floating-point types	float	

Variable Initialization & Assignment

- int x; // this is a variable definition also known as variable initialization
- x = 5; // assign the value 5 to variable x
- int x = 5; // initialize variable x with the value 5
- L-value – x: has an address in memory
- R-value – 5: value assigned to l-value

Variable Initialization & Assignment

- three ways to initialize variables.
  - c-like initialization
    - type variable = value;
  - constructor initialization
    - type variable (value);
  - uniform initialization (c++ 11)
    - type variable {value};
  - Examples:
    - int a = 5; // c- like
    - int b(3); // constructor
    - int c{ 2 }; // uniform

Keywords Const & Volatile

- Constant Variables : A read only variable that retains its value through out the program.

EXAMPLE: const double pi = 3.1415947;

- Volatile Variables: such variables can be modified not only by the program but also by other programs and external events. Events can be initiated by interrupts or by a hardware clock

EXAMPLE: volatile unsigned long clock ticks;

Global Variables

- used throughout the lifetime of the program
- must be declared outside main()
- If only declared, they can be assigned different values at different times in program lifetime
- storage allocated for the entire duration of the program
- known as static storage
- Variables with static storage, if not initialized, are automatically initialized to zero.

Local variables

- Exist only within the block in which it’s declared
- storage is known as automatic storage and only available during the lifetime of the block
- Variables with automatic storage are not implicitly initialized

Special Types of Variables

- Final - Once initialized, its value can’t be changed.
- final int i = 10;
- Static - These variables holds their value between function calls.
- static int y = 20;

Pointers

- a variable whose value is the address of another variable.

SYNTAX

type *var-name;

Example:

int *ip; // pointer to an integer

double *dp; // pointer to a double

float *fp; // pointer to a float

char *ch // pointer to a character

Type Deduction & Conversion

Type Deduction

- Auto

Int a = 0;

auto b = a; // the same as int b=
a

- Decltype

Int a =0;

Decltype(a) b; //same as int b

Type Deduction & Conversion

Type Conversions

- Implicit Type Conversion: Implicit conversions are automatically performed when a value is copied to a compatible type.

Example: short a = 2000;

int b;

b = a;

conversion from smaller datatype to a larger one - promotion

conversion from larger datatype to a smaller one - demotion

Explicit Type Conversion:

Done using type-casting.

2 types of Generic type-casting

- C-like
- Functional

Example:

double f = 3.14;

unsigned int n1 = (unsigned int)f; // C-style cast

unsigned int n2 = unsigned(f); // functional cast

BASIC I/O OPERATIONS

The stream classes

- istream class- deals with input streams
- ostream class- deals with output streams
- iostream class - handles both input and output, allowing bidirectional I/O.
- left shift operator (<<)- called stream insertion operator
- right shift operator (>>)- stream extraction operator

Basic I/O Operations

stream objects

- cin: standard input stream

Example:

```
cin >> name;
cin >> name >> age;
```

- cout: standard output stream

Example:

```
cout <<" Hello World ";
cout <<”Value of str is : “<< str
<< endl;
```

cerr: standard error (output) stream

Example:

```
cerr <<”Error message : “
cerr <<”Error message : “<<“Unable
to read....”<< endl;
```

clog: standard logging (output) stream

Example:

```
clog <<”Error message : “;
clog <<”Error message : “<<“Unable
to read....”<< endl;
```

Functions

Library Function

- built-in functions in C++ programming
- Mathematical Functions- header file <cmath>

Example:

log(x) logarithm of x

log 10(x) Logarithm of number x to the base 10

sqrt(x) Square root of x

pow(x, y) x raised to the power y

Character Functions- header file <cctype>

Example:

islower(c) It returns True if C is a lowercase letter otherwise False.

isupper(c) It returns True if C is an uppercase letter otherwise False.

toupper(c) It converts c to uppercase letter.

tolower(c) It converts c to lowercase letter.

Functions

User-defined Function

- groups code to perform a specific task

Syntax:

return type function name(parameter list)

{

body of the function

}

- Function Prototype- Function blueprint

Function Example:

```
/* Function definition */
int add(int a, int b) // Function declarator
{
int add;
add = a + b;
return add; // Return statement
}
```

Function Prototype Example:

```
int add(int, int);
int add(int a, int b);
```

Operators

- special type of functions that take one or more arguments and produce a new value
- Assignment Operator(=): copies rvalue into lvalue.
- Mathematical Operators (+,-,/,*,%): perform basic mathematical operations
- Shorthand notation to perform an operation and assignment at same time

Example,

int x = 10;

x += 4 // will add 4 to 10, and hence assign 14 to X.

x -= 5 // will subtract 5 from 10 and assign 5 to x.

Relational Operators (<,>,<=,>=,==,!=) : establish a relationship between operands

Logical Operators (&&,||): used to combine two different expressions together.

Operators

- Bitwise Operators (&,|,^,~): works only with integral data types like char, int and long
- Shift Operators (<<,>>) : used to shift bits of any variable
- Unary Operators (++,--): work on only one operand
- Ternary Operator (?:) : operator which has three operands

Comma Operator (,): used to separate variable names & expressions

Example:

int a, b, c; // variables declaration using comma operator

a = b++, c++; // a = c++ will be done.

Operators

Bitwise Operators

unsigned int a = 60; // 60 = 0011 1100

unsigned int b = 13; // 13 = 0000 1101

int c = 0;

- c = a & b;

```
0011 1100 &
0000 1101 =
------------
0000 1100
```

- c = a | b;

```
0011 1100 |
0000 1101 =
------------
0011 1101
```

c = a ^ b;

```
0011 1100 ^
0000 1101 =
------------
0011 0001
```

c = ~a;

```
0011 1100 ~
-------------
1100 0011
```

Operators

Shift Operators

- c = a << 2; //<< shifts left and adds zeros at the right end.

```
0011 1100 <<2
---------------
1111 0000
```

- c = a >> 2; //right shift (>>) operator shifts bits to the right.

```
0011 1100 >>2
---------------
0000 1111
```

Operators

```
#include <iostream>
using namespace std;
int main()
{
unsigned int a = 60;  // 60 = 0011 1100
unsigned int b = 13;  // 13 = 0000 1101
int c = 0;
c = a & b;            // 12 = 0000 1100
cout << "Line 1 - Value of c is : " << c << endl;
c = a | b;            // 61 = 0011 1101
cout << "Line 2 - Value of c is: " << c << endl;
```

Operators

```
c = a ^ b;          // 49 = 0011 0001
cout << "Line 3 - Value of c is: " << c << endl;
c = ~a;             // -61 = 1100 0011
cout << "Line 4 - Value of c is: " << c << endl;
c = a << 2;         // 240 = 1111 0000
cout << "Line 5 - Value of c is: " << c << endl;
c = a >> 2;         // 15 = 0000 1111
cout << "Line 6 - Value of c is: " << c << endl;
return 0;
}
```

Function Templates

- SYNTAX

template <template-parameters> function-declaration

- SYNTAX for calling template function

name <template-arguments> (function-arguments);

Function Templates

Example:

```
// the template prefix:
template<class T>
void swapVariables(T& var1, T&
var2)
{
  T temp;
  temp = var1;
  var1 = var2;
  var2 = temp;
}
```

```
int main( )
{
  float float1 = 3.5, float2 = 5.6;
 //call to the function:
 swapVariables(float1, float2);
}
```

Function Templates

```
// function template
#include<iostream>
using namespace std;
template<class T>
T sum(T a, T b)
{
T result;
result = a + b;
return result;
}
```

```
int main() {
int i = 5, j = 6, k;
double f = 2.0, g = 0.5, h;
k = sum<int>(i, j);
h = sum<double>(f, g);
cout << k <<'\n';
cout << h <<'\n';
return 0;
}
```

NAME VISIBILITY

- Variables, functions, and compound types need to be declared before being used
- An entity declared outside any block has global scope
- Entity declared within a block has block scope.
- In each scope, a name can only represent one entity
- Global entities cause possibility for name collision.
- This problem can be overcome using namespaces

NAME VISIBILITY

```
// inner block scopes
#include<iostream>
using namespace std;
int main()
{
int x = 10;
int y = 20;
{
int x; // ok, inner scope.
x = 50; // sets value to inner x
y = 50; // sets value to (outer) y
cout <<”inner block:\n”;
cout <<”x: “<< x <<’\n’;
cout <<”y: “<< y <<’\n’;
}
cout <<”outer block:\n”;
cout <<”x: “<< x <<’\n’;
cout <<”y: “<< y <<’\n’;
return 0;
}
```

Namespace

- SYNTAX

namespace identifier

{

named_entities

}

- identifier is any valid identifier
- named entities is the set of variables, types and functions
- To access these entities from outside the namespace, use scope operator "::“
- keyword “*using*” introduces a name into the current declarative region

Namespace

```
// namespaces
#include<iostream>
using namespace std;
namespace foo
{
int value() { return 5; }
}
namespace bar
{
const double pi = 3.1416;
double value() { return 2 * pi; }
}
int main() {
cout << foo::value() <<'\n';
cout << bar::value() <<'\n';
cout << bar::pi <<'\n';
return 0;
}
```

Namespace

```
#include<iostream>
using namespace std;
namespace first
{
int x = 5;
int y = 10;
}
namespace second
{
double x = 3.1416;
double y = 2.7183;
}
int main() {
using first::x;
using second::y;
cout << x <<’\n’;
cout << y <<’\n’;
cout << first::y <<’\n’;
cout << second::x <<’\n’;
return 0;
}
```

Namespace

```
#include<iostream>
using namespace std;
namespace first
{
int x = 5;
int y = 10;
}
namespace second
{
double x = 3.1416;
double y = 2.7183;
}
int main() {
using namespace first;
cout << x <<’\n’;
cout << y <<’\n’;
cout << second::x <<’\n’;
cout << second::y <<’\n’;
return 0;
}
```

EXERCISES

- Write a C++ program that outputs the following text on screen:

Oh what

a happy day!

Oh yes,

what a happy day!

Use the manipulator endl where appropriate.

- Write a program to add an 8% sales tax to a given amount and round the result to the nearest rupee.

- Write a program to tell whether a number is prime.

- Write a complete program that reads an number from the user, doubles it using user defined function DoubleNumber(),that takes one number (integer or float) and returns twice the value passed in, and then prints the doubled value out to the console.