THE UNIVERSITY OF UTAH
ASIA CAMPUS
SOUTH KOREA

Department of Electrical and Computer Engineering
Spring 2022

Accelerated Object Oriented Programming (CS 1420)

2. C++ Basics

Habtamu Minassie
Habtamu.aycheh@Utah.edu

Simple C++ Program Structure

The // characters begin a comment

Includes header file iostream library that defines input/output. **#include directives** are enclosed in angle brackets (< …>), it refers to a part of the C++ library called a standard header files

The "entry point" of our program.

```
// C++ program structure

#include<iostream>

int main()

{
std::cout<<"Welcome to C++\n";
return 0;

}
```

Signify the start and end of a series of statements (block of code)

Semicolon(;) shows end of statement

"\n" is an escape character that means "newline".

the program executed successfully (optional)

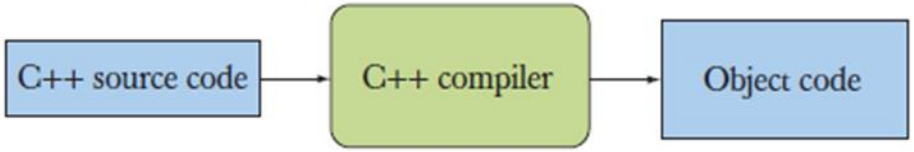
C++ Compilers

- **MinGW (Minimalist GNU for Windows)**
  - It's an opensource tool with no third-party requirements and works well with the development of Microsoft windows. It has GCC/g++ compilers to include C, C++ language compilers. Among many other tools, this compiler is liked most by the user due to the high level of portability available in GCC by ANSI Compliance.
- **Clang**
  - **Clang** compiler is preferred to be easily understandable compiler provided with front-end with more fast and reduced memory adapted with a Berkeley Software Distribution(BSD) license. A good feature of Clang is its GCC compatibility and its design is based on Low Level Virtual Machine (LLVM). It is built for a better analysis of the code with faster compilation
- **Visual C++**
  - A key feature of Visual C++ is the development of Microsoft Foundation Class library (MFC) architecture which provides the fastest executables, developing windows-based applications

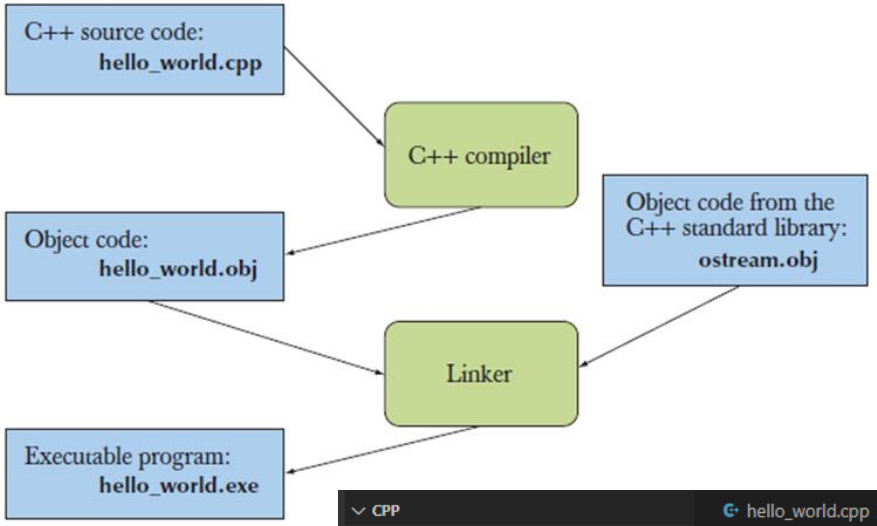
Example : The first C++ program

```cpp

// hello_world.cpp
// This program outputs the message “Hello World!” to the monitor
#include <iostream>
int main() // C++ programs start by executing the function main
{
std::cout << "Hello World"; // output “Hello World”
return 0;
}

```

Compilation



Linking



Compilation and Linking Demo

- g++ or clang compiler- using command line

Command Prompt

```
               2 Dir(s)  155,738,722,304 bytes free

D:\CPP>g++ --version
g++ (GCC) 11.2.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


D:\CPP>g++ -c -o hello_world.obj hello_world.cpp

D:\CPP>g++  -o hello_world.exe hello_world.obj

D:\CPP>hello_world
Hello C++
D:\CPP>dir
 Volume in drive D is Data
 Volume Serial Number is A8AC-BCB9

 Directory of D:\CPP

02/04/2022  05:10 PM    <DIR>          .
02/04/2022  05:10 PM    <DIR>          ..
02/04/2022  12:07 PM               351 hello_world.cpp
02/04/2022  05:10 PM         2,965,139 hello_world.exe
02/04/2022  05:10 PM             1,839 hello_world.obj
               3 File(s)      2,967,329 bytes
               2 Dir(s)  155,735,752,704 bytes free
```

Developer Command Prompt for VS 2022

```
D:\CPP>clang --version
clang version 12.0.0
Target: i686-pc-windows-msvc
Thread model: posix
InstalledDir: C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\Llvm\bin

D:\CPP>clang -c -o hello_world.obj hello_world.cpp

D:\CPP>clang  -o hello_world.exe hello_world.obj

D:\CPP>hello_world.exe
Hello C++
D:\CPP>dir
 Volume in drive D is Data
 Volume Serial Number is A8AC-BCB9

 Directory of D:\CPP

02/10/2022  04:12 PM    <DIR>          .
02/10/2022  04:12 PM    <DIR>          ..
02/09/2022  05:36 PM               351 hello_world.cpp
02/10/2022  04:12 PM           181,760 hello_world.exe
02/10/2022  04:12 PM            62,314 hello_world.obj
               3 File(s)        244,425 bytes
               2 Dir(s)  155,707,822,080 bytes free

D:\CPP>
```

Windows PowerShell

```
PS D:\CPP>
PS D:\CPP> g++ -c -o hello_world.obj hello_world.cpp
PS D:\CPP> g++ -o hello_world  hello_world.obj
PS D:\CPP> ./hello_world
Hello World
PS D:\CPP>
```

Header files: #include <…>

- We need header files to add or include predefined libraries to our C/C++ program
- Header files contain definitions of functions and variables
- In C/C++ header files are imported by using the pre-processor #include<…> statement.
- C header files have an extension of ".h"
- Note that All C code is valid C++ code
- Example
  - #Include<iostream>
    - Tells the preprocessor to include standard input/output streams like cout and cin
- The preprocessors are the directives, which give instructions to the compiler to preprocess the information before actual compilation starts
- Preprocessor directives are not statements, so they do not end in a semicolon (;).

Output

- std::cout is an object of the output stream that is used to display results on standard output screen
- std::cout uses the extraction operator(<<).

```
// Example
// Text-printing program.
#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main()
{
   std::cout << "Welcome to C++!\n"; // display message

   return 0; // indicate that program ended successfully
} // end function main
```

```
Welcome to C++!
```

\n (new line) escape character

- Each time the \n (newline) is encountered, the screen cursor is positioned to the beginning of the next line

```
// Example
// Printing multiple lines of text with a single statement.
#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main()
{
   std::cout << "Welcome\nto\n\nC++!\n";
} // end function main
```

```
Welcome
to

C++!
```

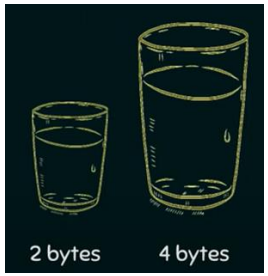
Input

- cin is an object of the input stream and is used to take input from standard keyboard
- cin uses the insertion operator(>>)

```
// Example
// Addition program that displays the sum of two integers.
#include <iostream> // allows program to perform input and output

// function main begins program execution
int main()
{
   // variable declarations
   int number1; // first integer to add
   int number2; // second integer to add
   int sum; // sum of number1 and number2

   std::cout << "Enter first integer: "; // prompt user for data
   std::cin >> number1; // read first integer from user into number1

   std::cout << "Enter second integer: "; // prompt user for data
   std::cin >> number2; // read second integer from user into number2

   sum = number1 + number2; // add the numbers; store result in sum

   std::cout << "Sum is " << sum << std::endl; // display sum; end line
} // end function main
```

Variables and Data Types

- One feature present in all computer languages is the identifier. Identifiers allow us **to name data** and other objects in the program. Each identified object in the computer is stored at a unique address.
- Variables are named memory locations.
  - Every variable has a name, a type, a size and a value
- A data type defines a set of values and a set of operations that can be applied on those values.
  - Types are one of the most fundamental concepts in programming



2 bytes = 16 bits

4 bytes = 32 bits

More the size, more content it can hold.

Example

Value

```
int number = 20;
```

Data Type

Variable Name (identifier)

sizeof(int)

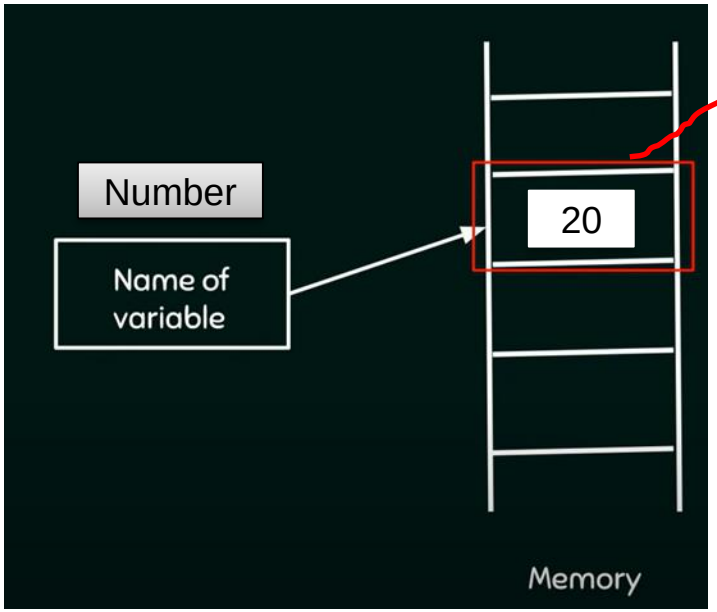
4 bytes = 32 bits

0001 0100

0000 0000

0000 0000

0000 0000

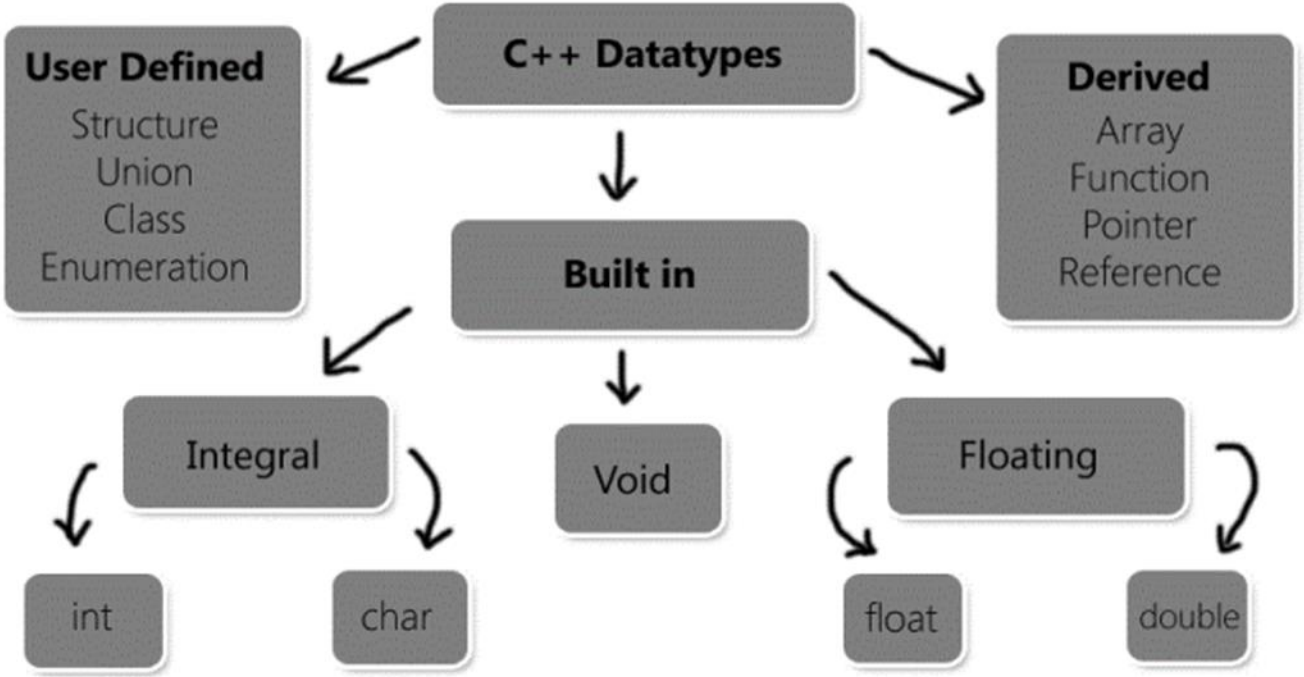


Fundamental Data Types in C++

- Fundamental (also called primary or primitive) data types are the basic built-in or predefined data types that we can directly use in our programs.
- A data type tells a variable the kind and size of data it can store
- When we declare a variable, the compiler allocates memory for it on the basis of its data type.

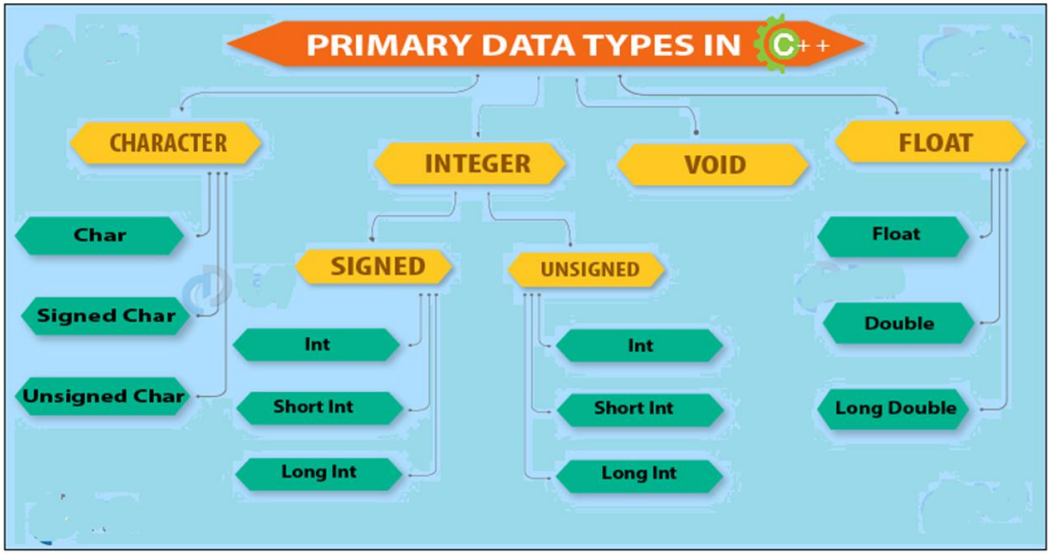
primitive data types	
Type	**Description**
bool	True or false values.
char	Single octet (one byte) value
int	Integer values
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

Categories of C++ Data types



Type Modifiers

- C++ primitive data types can be modified using one or more types of suitable modifiers :
  - singed
  - unsinged
  - short
  - long



Size of C++ data types – sizeof(…)

https://en.cppreference.com/w/cpp/language/types

Data type	Size (in Bytes)	Description	Example
signed int / int	4	Stores integers values starting from -2,147,483,648 to 2,147,483,647	signed int x = -40;
unsigned int	4	Stores 0 and positive integers (0 to 4,294,967,295)	unsigned int x = 40;
short / signed short	2	Equivalent to short int or signed short int, stores small integers ranging from -32768 to 32767	short x = -2;
unsigned short	2	Equivalent to unsigned short int, stores 0 and small positive integers ranging from 0 to 65535	unsigned short x = 2;
long	4	Equivalent to long int, stores large integers	long x= 4356;
unsigned long	4	Equivalent to unsigned long int, stores 0 and large positive integers	unsigned long x = 562;
long long	8	Equivalent to long long int, stores very large integers	long long x= -243568;
unsigned long long	8	Equivalent to unsigned long long int, stores 0 and very large positive integers	unsigned long long x = 12459;
long double	16	Stores large floating-point values	long double x = 432.6781;
signed char / char	1	Stores characters ranging from -128 to 127	signed char ch = ‘b’;
unsigned char	1	Stores characters ranging from 0 to 255	unsigned char ch = ‘g’;

Size of data type …

- The size of a data type depends on system language and a machine architecture.
  - For example, in C/C++, the size of a long integer is 4 bytes (32-bits) on a 32-bit machine, and 8 bytes (64-bit) on a 64-bit machine in relation to the word size (register size) of the machine architecture.

- **sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(long) ≤ sizeof(long long)**
- **1 ≤ sizeof(bool) ≤ sizeof(long)**
- **sizeof(char) ≤ sizeof(wchar_t) ≤ sizeof(long)**
- **sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)**

C++ Variable Naming Rules

- A variable name (identifier) consists of a sequence of letters and digits.
  - The first character must be a letter
  - Variable name cannot contain special characters such as comma, semicolon, and white space, however, we can have a underscore (symbol _) as a part of the variable name.
  - C++ imposes no limit on the number of characters in a name.
  - A C++ keyword cannot be used as a name of a user-defined entity

Note that:

- Names can contain **$** and _. But allowing the character $ in a name may yield nonportable programs so that it is advised not to use it. It is also recommended not to use _ at the beginning of a variable name.
- C++ is case sensitive—uppercase and lowercase letters are different, so a1 and A1 are different identifiers.
- Good naming is always fundamental to effective code reviews and collaboration
  - Choosing meaningful identifiers makes a program self-explanatory—a person can understand the program simply by reading it rather than having to refer to the manuals or comments

Examples

Identifier	Remark
roll_no	✓
sum	✓
Data12	✓
Student's	✗
8var	✗
Roll no	✗
012	✗
if	✗
Pay.due	✗
.name	✗
class	✗
CLASS	✓
u_name	✓
_class	✓
bAr	✓
DEFINED	✓

this_is_a_most_unusually_long_identifier_that_is_better_avoided ✓

Multiword Delimited

This co**n**vention is to separate words in a variable name without the use of whitespace

Snakecase: Words are delimited by an underscore.

- Variable_one
- Variable_two

PascalCase: Words are delimited by capital letters

- VariableOne
- VariableTwo

Camelcase: Words are delimited by capital letters, except the initial word.

- variableOne
- variableTwo

List of Keywords in C/C++

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	false
friend	inline	mutable	namespace	new
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

Variable declaration

- C++ is a strongly-typed language
- All variables must be declared with a name and a data type before they can be used in a program.
- Declarations of variables can be placed almost anywhere in a program, but they must appear before their corresponding variables are used in the program

```
char c;      //character variable declaration.
int area;    //integer variable declaration.
float num;   //float variable declaration.
```

```
int a;  //integer variable declaration.
int b;  //integer variable declaration.
int c;  //integer variable declaration.
```

```
int a, b, c;    //more than one variable declaration.
```

Initialization of variables

- When the variables are declared, they have an undetermined or garbage value until they are assigned a value for the first time
- There are 4 basic ways to initialize variables in C++

```
// C++ variable initialization
int num1; // no initializer --> Default initialization
int num2 = 5; // initializer after equals sign-->Copy initialization
int num3( 6 ); // initializer in parenthesis-->Direct initialization
int num4 { 7 }; // initializer in braces -->Brace initialization
```

```
/*Copy initialization --> C language style
    When an initializer is provided after an equals sign, it is called copy initialization.*/
int width = 5; // copy initialization of value 5 into variable width
/*Direct initialization-
    When an initializer is provided inside parenthesis, it is called direct initialization.*/
/*Direct initialization
    When an initializer is provided inside parenthesis, it is called direct initialization*/
int width( 5 ); // direct initialization of value 5 into variable width
/* For simple data types (like int), copy and direct initialization are essentially the same.
For more complicated types, direct initialization tends to be
more efficient than copy initialization*/
```

Brace initialization

- Unfortunately, direct initialization can't be used for all types of initialization (such as initializing an object with a list of data).
- To provide a more consistent initialization mechanism, there's brace initialization (also called uniform initialization or list initialization) that use curly braces.
- **Brace initialization comes in three forms:**

```
int width { 5 }; // direct brace initialization of value 5 into variable width (preferred)
int height = { 6 }; // copy brace initialization of value 6 into variable height
int depth {}; // value initialization will initialize the variable to zero
```

- Brace initialization has the added benefit of disallowing "narrowing" conversions. This means that if you try to use brace initialization to initialize a variable with a value it can not safely hold, the compiler will throw a warning or an error. For example:

```
int width { 4.5 }; // error: not all double values fit into an int
```

Best practice: Use Brace initialization in c++ for variable initialization

Example

```
#include <iostream>
int main()
{
    int sum1,sum2; //Variable declaration
    int num1{5}; // variable declaration and initialization
    int num2(10); // variable declaration and initialization
    int num3; // No initialization
    sum1=num1+num2;
    sum2=num1+num3;
    std::cout<<num1<<" + " <<num2<<" = "<<sum1<<endl;
    std::cout<<"SUM2 = "<<sum2<<std::endl; //prints some garbage value

return 0;
}
```

Output

```
5 + 10 = 15
SUM2 = -800058475
```

Local and Global Scopes

- C++ statements are enclosed in curly braces ({...})
- Variables declared within a block are called local variables and only accessible from within the block
- Variable declared outside of any block called global variables and accessible from everywhere in the program.

```
#include<iostream>
using namespace std;
int x=10,y =10; //  x  and y are global variables
int main()
{
    int a=7,b=2; //x  and y are Local variables
    int x =5,y=5; //a  and b are local variables
    int sum1 = x+y+a+b;
    int sum2 = ::x + ::y + a + b; // The :: operator is scope resolution operator
    cout<<"Sum1 ="<<sum1<<endl;
    cout<<"Sum2 ="<<sum2<<endl;
}
```

Namespaces

- Global variables present many problems in large software systems because they can be accessed and possibly modified any where in the program => Name conflict
- namespace is a mechanism that allows a group of related names to be defined in one place.

```
#include<iostream>
using namespace std; // Standard Name space
namespace namespace1 // user defined namespace1
{
    int x=4,y=0;
}
namespace namespace2 // user defined namespace1
{
    int x=4,y=0;
}
int main()
{
    int x=20; //Local variables
    cout<<"x from namespace1 ="<<namespace1::x<<endl;
    cout<<"y from namespace1 ="<<namespace2::y<<endl;
    cout<<"x from namespace2 ="<<namespace2::x<<endl;
    cout<<"y from namespace2 ="<<namespace2::y<<endl;
    cout<<"Local x ="<<x<<endl;
}
```

The using Keyword makes just std namespace accessible

```
using namespace std;
```

```
using std::cout,std::cin,std::endl;
```

```
std::cout<<"...";
```

C++ Literals and Constants

- C++ literals are used for representing fixed values.
- The values assigned to each constant variables are referred to as the literals

```
int a=3; //Integer Literals
char ch = 'A'; //character literal
double pi = 3.14; //Floating-point Literal
bool isEmpy = true; // Boolean literal
```

Literal suffixes

- If the default type of a literal is not as desired, you can change the type of a literal by adding a suffix:

Data Type	Suffix	Meaning
int	u or U	unsigned int
int	l or L	long
int	ul or LU	unsigned long
int	ll or LL	long long
int	ull or LLU	unsigned long long
double	f or F	float
double	l or L	long double

```
//Example
float num =2.74f;
unsigned val =12u;
```

Escape Sequences

- Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C++ programming. For example, newline (enter), tab, question mark, etc.
- In order to use these characters, escape sequences are used.

Escape Sequences	Characters
\n	Newline
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark

C++ Constants

- In C++, we can create variables whose value cannot be changed. For that, we use the const keyword.

```
//Example
const int PI = 3.14;
const double MAX_VALUE = 2000.5;
PI= 3.142;  // error: assignment of read-only variable 'PI'
```

Type casting

- **Implicit type conversion**
  - the compiler can implicitly convert a value from one data type to another

```
double d = 10 / 4; // does integer division, initializes d with value 2.0
```


```
double d = 10 / 4; // does integer division, initializes d with value 2.0
```


```
int x { 10 };
int y { 4 };
double d = x / y; // does integer division, initializes d with value 2.0
```

Explicit type conversion

- C++ major type casting methods are:
  - C-style casts and
  - static casts

C-style casts

```
#include <iostream>

int main()
{
    int x { 10 };
    int y { 4 };


    double d { (double)x / y }; // convert x to a double so we get floating point division
    std::cout << d; // prints 2.5

    return 0;
}
```

static casts

- C++ introduces a casting operator called **static_cast**, which can be used to convert a value of one type to a value of another type.

```
int main()
{
    char c { 'a' };
    std::cout << c << ' ' << static_cast<int>(c) << '\n'; // prints a 97

    return 0;
}
```

```
#include <iostream>
int main()
{
    int x { 10 };
    int y { 4 };
    // static cast x to a double so we get floating point division
    double d { static_cast<double>(x) / y };
    std::cout << d; // prints 2.5
    return 0;
}
```

Best practice: Use static_cast when you need to convert a value from one type to another type.

Expressions

- An expression is a sequence of operators and their operands, that specifies a computation.

Assignment Operators

Assignment Operator	Shorthand operation
a = a + b	a += b
a = a - b	a -= b
a = a * b	a *= b
a = a / b	a /= b
a = a % b	a %= b

```cpp
// Example
#include <iostream>
int main()
{
    int a=10;
    int c=a+=5;

    std::cout<<"c = "<<c <<endl;
return 0;
}
```

```
output = 15
```

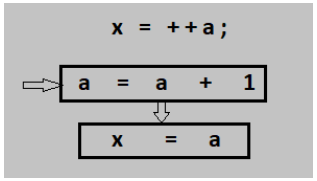
Arithmetic Expressions

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	-	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

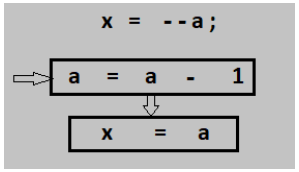
C++ Increment and decrement operator

Operator	Meaning
++	Increment Operator
--	Decrement Operator

Pre-increment

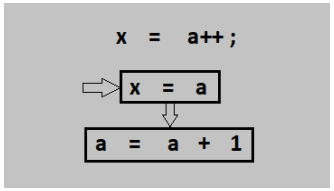


Pre-decrement

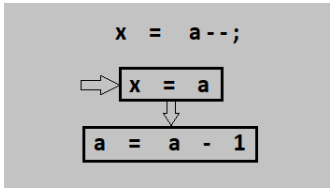


First, the value of the variable a incremented by 1 (++a) or decremented by 1(--a) and store in the memory location of variable **a**. Second, the value of variable **a** will be assigned to the variable **x**

Post-increment



Post-decrement



First, the value of the variable a will assign to the variable x. Second, the value of the variable a will be incremented by 1(a++) or decremented(a--) and store in the memory location of the variable a.

Example

```cpp
// Example  1
#include <iostream>
int main()
{
    int a=5;
    int x=++a;
    std::cout<<a<<endl;
    std::cout<<x<<endl;
return 0;
}
```

```
6
6
```

```cpp
// Example  2
#include <iostream>
int main()
{
    int a=5;
    int x=--a;
    std::cout<<a<<endl;
    std::cout<<x<<endl;
return 0;
}
```

```
4
4
```

```cpp
// Example  5
#include <iostream>
int main()
{
    int a=5;
    int x=a;
    std::cout<<a++<<endl;
    std::cout<<a<<endl;
    std::cout<<++a<<endl;
    std::cout<<a<<endl;
return 0;
}
```

```
5
6
7
7
```

```cpp
// Example  3
#include <iostream>
int main()
{
    int a=5;
    int x=a++;
    std::cout<<a<<endl;
    std::cout<<x<<endl;
return 0;
}
```

```
6
5
```

```cpp
// Example  4
#include <iostream>
int main()
{
    int a=5;
    int x=a--;
    std::cout<<a<<endl;
    std::cout<<x<<endl;
return 0;
}
```

```
4
5
```

Memory Concepts

- Variable names correspond to locations in the computer's memory.
- Every variable has a name, a type, a size and a value.
- When a value is placed in a memory location, it overwrites the previous value in that location

```
int number1 {45};
int number2 {72};
int sum = number1 + number2;
```

number1	45
number2	72
sum	117

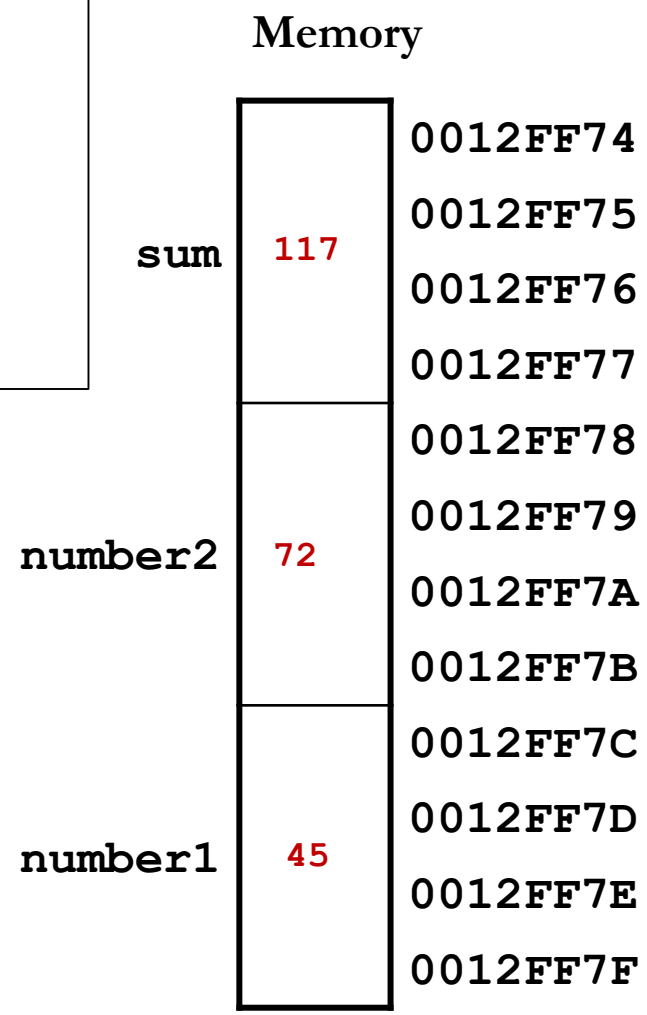
```
number1 = 50;
sum = number1 + number2;
```

number1	50
number2	72
sum	120

Illustration

```
int number1, number2, number3;
std::cout << "Enter first integer: ";
std::cin >> number1;
std::cout << "Enter second integer: ";
std::cin >> number2;
sum = number1 + number2;
std::cout <<"Sum is"<< sum << std::endl;
```

```
Enter first integer: 45
Enter second integer: 72
Sum is = 117
```



Memory is Limited

Open gallery

Surface Laptop 3 - 13.5", Black (metal), Intel Core i5, 8GB, 256GB

Wish list

Slim and stylish, available in 13.5" and 15" touchscreens, rich color options,[1] and two durable finishes. Make a powerful statement and get improved speed, performance, and all-day battery life.[2]

The Microsoft Store Promise for Surface
Shop with confidence at Microsoft Store. We're offering 60-day returns on Surface products, plus free digital workshops, remote learning opportunities, and more to help you get the most from your new device.*
Learn more >

Bundle and save with the Surface Laptop 3 Essentials Bundle
Includes your choice of Surface Laptop 3, Microsoft 365 and Microsoft Complete Protection Plan.
Build your bundle >

This literally means your program can run up to **8 GB** memory

Value Representation

- **Binary, Octal, Decimal, Hex**
  - The six letters (in addition to the 10 integers) in hexadecimal represent: 10 (A), 11 (B), 12 (C), 13 (D), 14 (E), and 15 (F), respectively.

Name	Radix	Digits
Binary	**2**	**0,1**
Octal	**8**	**0,1,2,3,4,5,6,7**
Decimal	**10**	**0,1,2,3,4,5,6,7,8,9**
Hexadecimal	**16**	**0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F**

```
// integer Literal
int num;
num = 56; // Decimal
num = 0b00111000; // Binary-->0b or 0B followed by one or more binary digits
num = 070; // Octal-->0 followed by zero or more octal digits
num = 0x38; // Hexadecimal-->0x or 0X followed by one or more hexadecimal digits
```

LAB 2: Arithmetic operations

- **Write a c++ program that declare three floating numbers a=2, b=3, and c=4**
  - Store the summation of a, b, c in variable x
  - What is the value of std::cout << b/a?
  - What is the value of std::cout << b/c?
  - How about changing the type from float to int?
- **Write a C++ program that converts degrees Fahrenheit to Celsius, which is given by the following formula:**

$$T_C = \left(T_F - 32\right) \cdot \frac{5}{9}$$

  - Get the output in the following format:

```
cout << TF << " Fahrenheit = " << TC << " Celsius" << endl;
```