



CS1002 – Programming Fundamentals

Lecture # 09
Monday, September 19, 2022
FALL 2022
FAST – NUCES, Faisalabad Campus


Muhammad Yousaf

General form of a C++ program

```
// Program description
#include directives
global declarations
int main()
{
    constant declarations
    variable declarations
    executable statements
    return 0;
}
```

Why `int main () ...?`

- `void` means that the function has no return-value, so you don't need `return`
- `int` -- which is also implied for functions with no stated return-type -- means that the function should return an integer, so you must use `return`
- The return-value of `main()` is also the return-value for the program, and its value is often used to tell the operating-system if the program ran successfully or not. Typically a return-value of **0 (zero)** tells that the program ran **without a problem**, while various **non-zero values** indicates **different problems** (e.g. **1=couldn't read a file**, **2=bad arguments**, **3=overflow**). Some OSs define which values should be used with what type of errors, others let's the programmer choose what he'd like for his program
- You should use a `main()` of type `int`, and return a **value (zero)** at the **end of your program**. The revised language-definition for C++ makes `int` the only legal type for `main()` (many compilers will at least warn you otherwise)... For C, `void` as return-type for `main()` has never really been in use... So use "`int main()`" and "`return 0`".

- 
- When you use `void main()`, you're making a mistake. Please don't do it
 - C programming language: ISO 9899 paragraph 5.1.2.2.1.1 Program startup
 - The function called at program startup is named `main`. It shall be defined with a return type of `int` and with no parameters: `int main(void) { /* ... */ }` or with two parameters `int main(int argc, char *argv[]) { /* ... */ }` or equivalent.
 - C++ programming language : ISO 14882 paragraph 3.6.1 Main function
 - A program shall contain a global function called `main`, which is the designated start of the program [...] This function shall not be overloaded. It shall have a return type of type `int`, but otherwise its type is implementation-defined. All implementations shall allow both of the following definitions of `main`: `int main() { /* ... */ }` and `int main(int argc, char* argv[]) { /* ... */ }`

Includes

- The statement: `#include <iostream>` inserts the contents of the file `iostream` inside your file before the compiler starts
- Definitions that allow your program to use the functions and classes that make up the standard C++ library are in these files.
- You can include your own file(s):

```
#include "myfile.h"
```

C++ compiler directives

- Compiler directives appear in **blue** in Visual C++
- The **#include** directive tells the compiler to include some already existing C++ code in your program
- The included file is then linked with the program
- There are two forms of #include statements:

#include <iostream> //for pre-defined files //searches in the standard files

#include "my_lib.h" //for user-defined files //first search in current directory

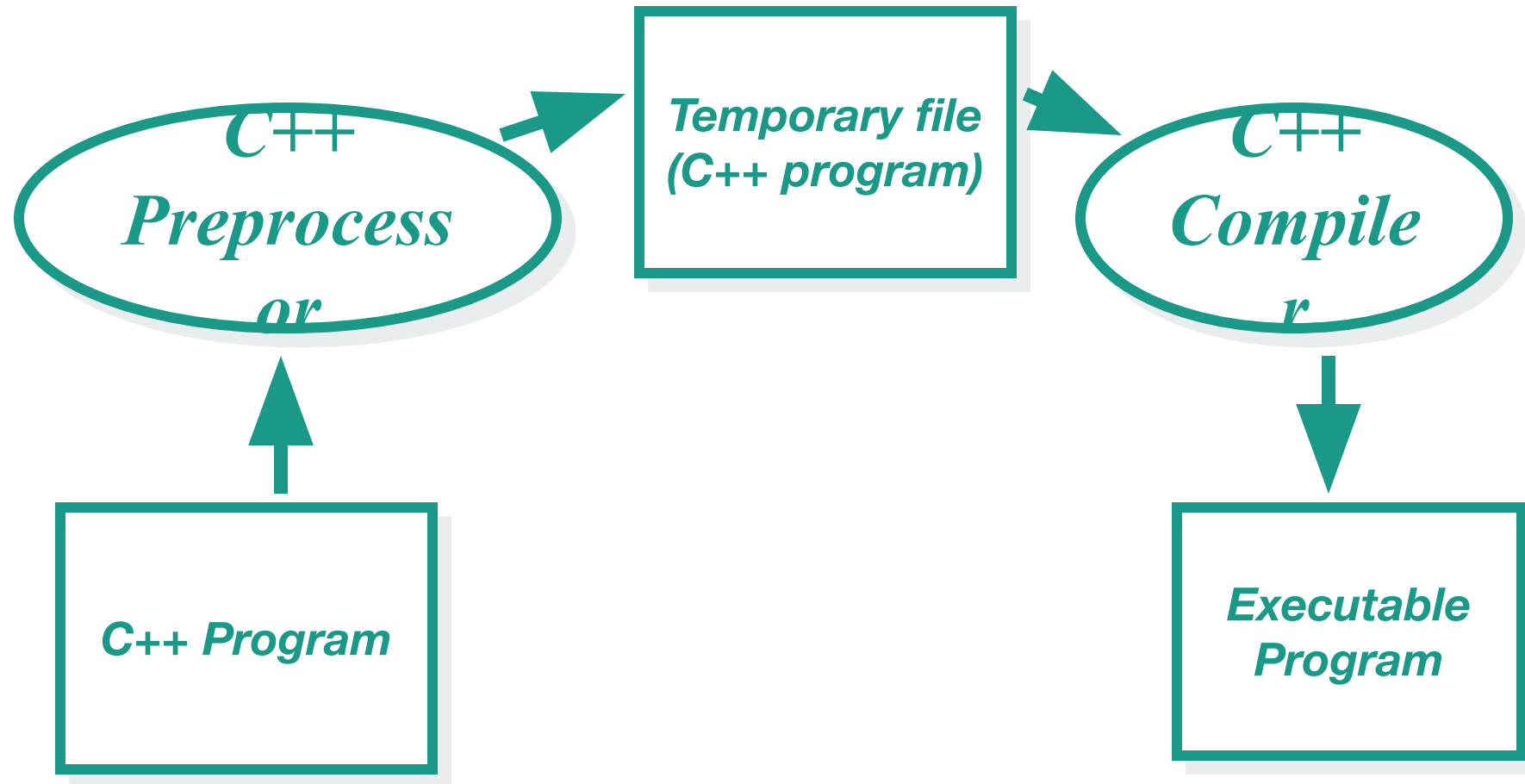
The Preprocessor

- All preprocessor directives begin with character '#'
- The Preprocessors are the directives, which give instruction to the compiler to preprocess the information before actual compilation starts.
- Preprocessor directives are not C++ statements so they do not end with semicolon (;)
- You already have seen `#include` directive in all examples. This macro is used to include a contents of a file into your source file.
 - `include`: replaced with contents of a file

C++ Preprocessor

- C++ Compilers **automatically invoke a preprocessor** that takes care of `#include` statements and some other special directives
- Definitions that **allow your program** to use the functions and classes that make up the standard C++ library are in these files
- You don't need to do anything special to run the preprocessor - it happens automatically

Preprocessing



Preprocessor Directives

- Preprocessor directives: Begin with `#`
- Processed before compiling
- `#include`, `#define`, `#if`, `#elif`, `#else`, `endif`, `#error`, `#line`, `#pragma`

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

#define ONE 1
#define TWO ONE + ONE
#define THREE TWO + ONE

int main()
{
    cout<<"Hi scientists"<<endl;
    cout<<THREE<<endl;
    return 0;
}
```

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

#define THREE 3
#define SIZE 10
#if SIZE > 5
    #undef THREE
    #define THREE 100
#endif

int main()
{
    cout<<"Hi scientists"<<endl;
    cout<<THREE<<endl;
    return 0;
}
```

Some common include statements

- Basic I/O: `iostream.h`
 - Provides functionality of input and output
- I/O manipulation: `iomanip.h`
 - Format's the input and output
- Standard Library: `stdlib.h`
 - Functions for memory allocation, process control, conversion etc.
- Time and Date support: `time.h`
 - Functionality of time manipulation
- Mathematics support: `math.h`
 - Functionality of basic mathematical functions

Today's Lecture

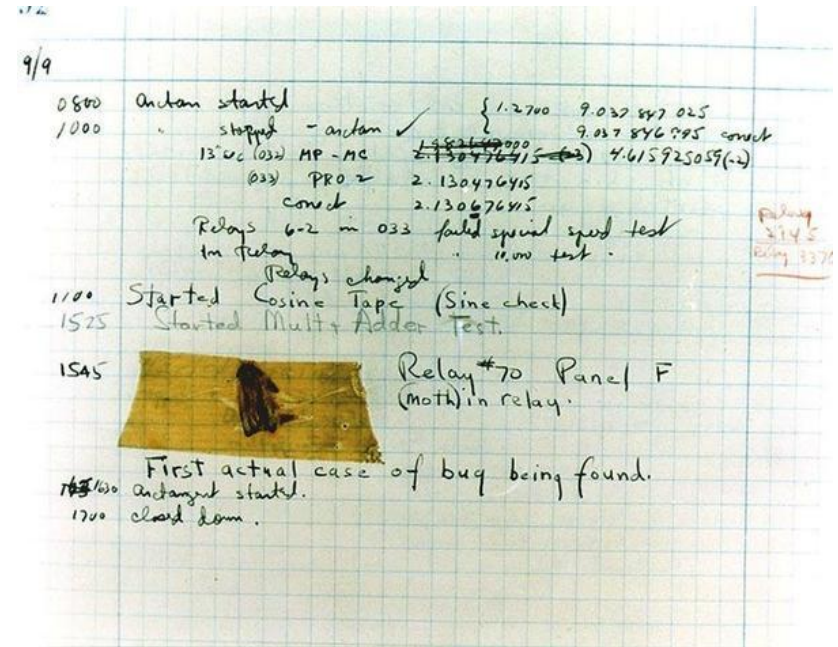
In this lecture, you will:

- Explore simple data types
- Discover how to use arithmetic operators
- Examine how a program evaluates arithmetic expressions
- Learn what an assignment statement is and what it does
- Become familiar with the string data type
- Become familiar with the use of increment and decrement operators
- Explore how to properly structure a program, including using comments to document a program
- Learn how to write a C++ program

Testing and debugging

- Bug
 - A logical mistake in a program
- Debugging
 - Eliminating mistakes in programs
 - Term used when a moth caused a failed relay on the Harvard Mark 1 computer. Grace Hopper and other programmers taped the moth in logbook stating:

“First actual case of a bug being found.”



Program errors

- **Syntax errors**

- Violation of the grammar rules of the language
- Discovered by the compiler
- Error messages may not always show correct location of errors

- **Run-time errors**

- Error conditions detected by the computer at run-time

- **Logic errors**

- Errors in the program's algorithm
- Most difficult to diagnose
- Computer does not recognize as an error



Harry @HareebAbbasi · 7h

Consumption of health is injurious to liquor !

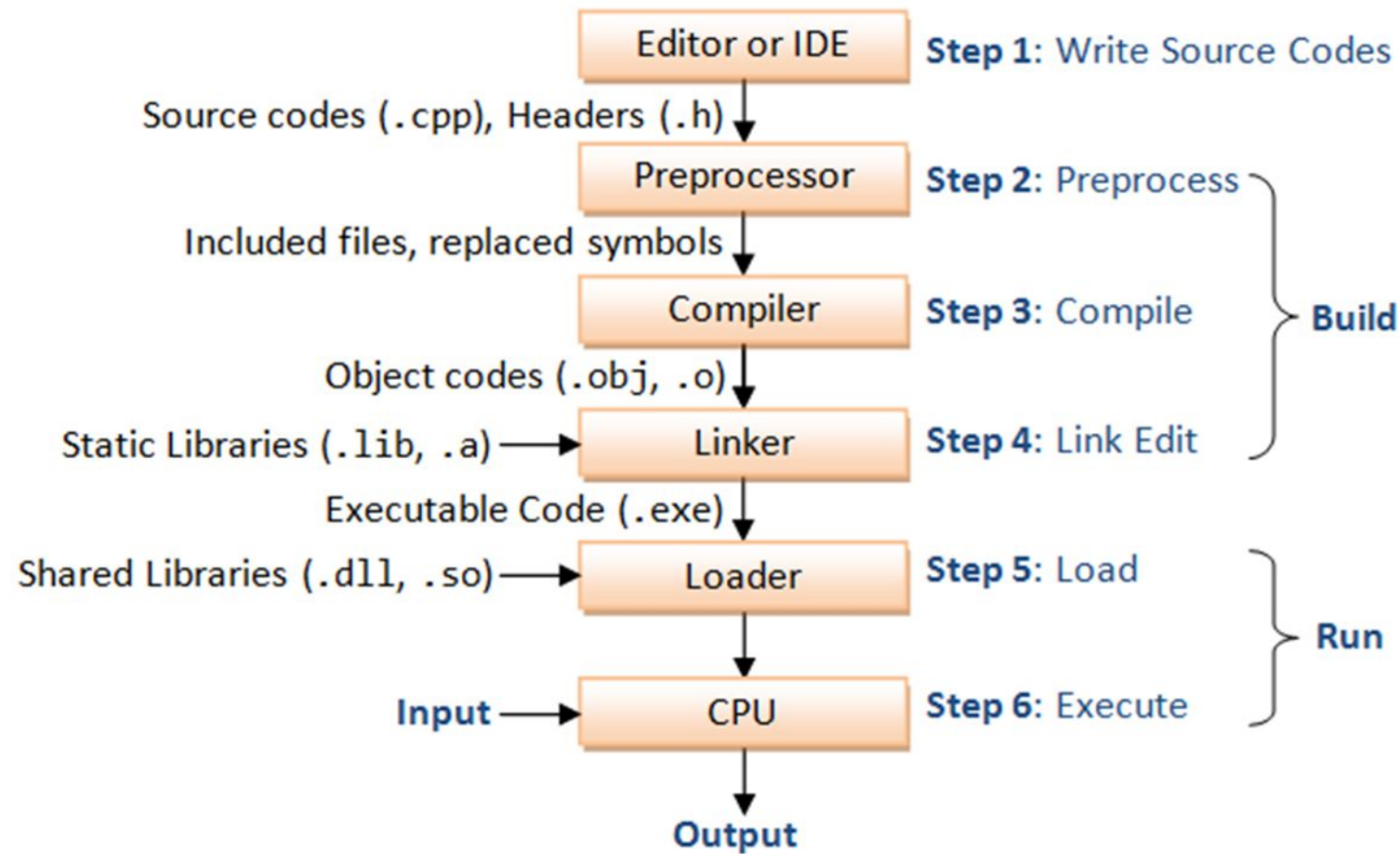
[#UmarAkmal](#)



What makes a bad program?

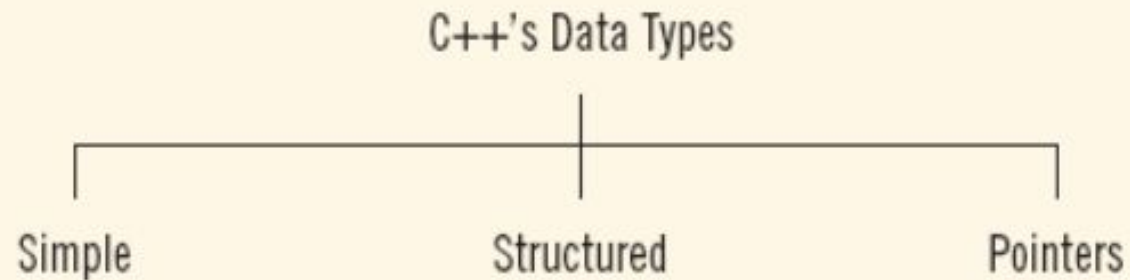
- Writing Code without detailed analysis and design
- Repeating trial and error without understanding the problem
- Debugging the program line by line, statement by statement
- Writing tricky and dirty programs

Live cycle of a program



Data Types

- **Data type:** Set of values together with a set of operations
- C++ data types fall into three categories:



Simple Data Types

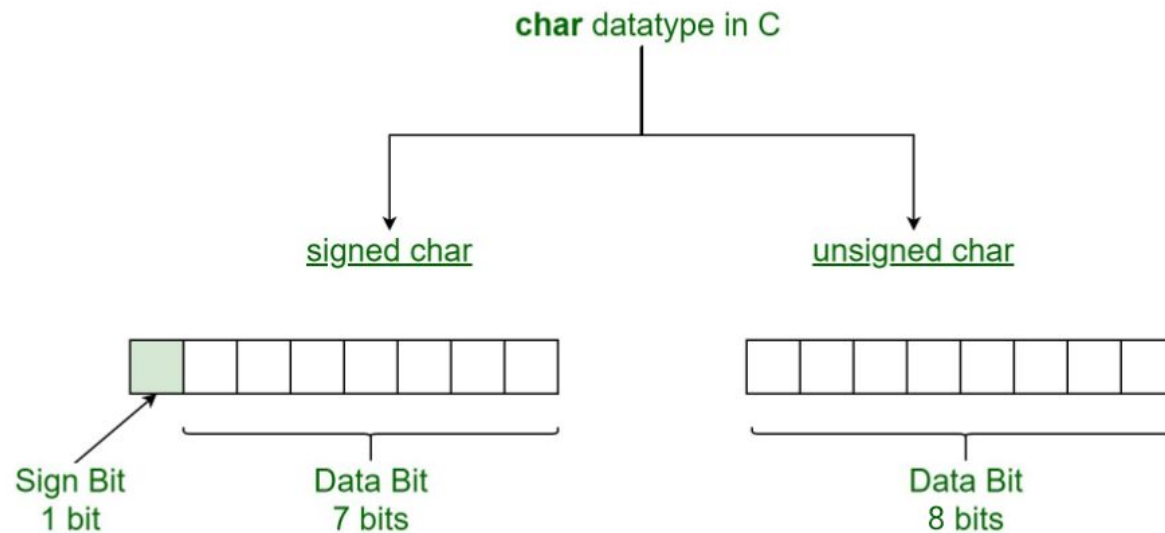
Three categories of simple data

- **Integral:** integers (numbers without a decimal)
- **Floating-point:** decimal numbers
- **Enumeration type:** user-defined data type

Simple Data Types (cont'd.)

Integral data types are further classified into nine categories:

- char, short, int, long, bool
- unsigned char, unsigned short, unsigned int, unsigned long



Simple Data Types (cont'd.)

- Different compilers may allow different ranges of values

Data Type	Values	Storage (in Bytes)
int	-2147483648 to 2147483647	4
bool	true and false	1
char	-128 to 127	1

int **Data Type**

Examples:

-6728

0

78

+763

- Positive integers do not need a + sign
- **No commas** are used within an integer [76,385]
 - Commas are used for separating items in a list



bool **Data Type**

- bool type
 - Two values: true and false
- Manipulate logical (Boolean) expressions
- true and false
 - Logical values
- bool, true, and false
 - Reserved words

char **Data Type**

- The smallest integral data type
- Used for characters: letters, digits, and special symbols
- Each character is enclosed in single quotes
 - 'A', 'a', '0', '*', '+', '\$', '&'
- A blank space is a character
 - Written ' ', with a space left between the single quotes
- 'abc' and '!=' are not char

Floating-Point Data Types

- You may be familiar with scientific notation. For example:
 - $43872918 = 4.3872918 * 10^7$ {10 to the power of seven}
 - $.0000265 = 2.65 * 10^{-5}$ {10 to the power of minus five}
 - $47.9832 = 4.79832 * 10^1$ {10 to the power of one}

Floating-Point Data Types

- C++ uses scientific notation to represent real numbers (floating-point notation)

Real Number	C++ Floating-Point Notation
75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

Floating-Point Data Types (cont'd.)

- **float:** represents any real number
 - Range: $-3.4\text{E}+38$ to $3.4\text{E}+38$ (four bytes)
- **double:** represents any real number
 - Range: $-1.7\text{E}+308$ to $1.7\text{E}+308$ (eight bytes)

Floating-Point Data Types (cont'd.)

- Maximum number of significant digits (decimal places) for float values is 6 or 7
- Maximum number of significant digits for double is 15
- **Precision:** Maximum number of significant digits
 - float values are called single precision
 - double values are called double precision

```
#include "iostream"
#include <iomanip>

int main()
{
    double d1 = 45.123456789101112131415;
    float f1 = 45.123456789101112131415;

    std::cout<<"Hi scientists"<<std::endl;
    std::cout<<std::setprecision(15) <<d1<<std::endl;
    std::cout<<std::setprecision(15)<<f1<<std::endl;
    return 0;
}
```

```
Hi scientists
45.1234567891011
45.1234550476074
```

Data type and variables

- When we declare a variable we not only specify the variable we also specify what type of data a variable can store. A syntax rule to declare a variable

`datatype identifier;`

- For example consider the following examples:

`int counter;`

`double interestRate;`

`char grade;`

Datatypes and their ranges

Type	Width	Common Range
char	8	-128 to 127
unsigned char	8	0 to 255
int	32	-2,147,483,648 to 2,147,483,647
unsigned int	32	0 to 4,294,967,295
short int	16	-32768 to 32767
unsigned short int	16	0 to 65535
long int	64	-2,147,483,648 to 2,147,483,647
unsigned long int	64	0 to 4,294,967,295
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 3.4E+4932

Arithmetic Operators and Operator Precedence

C++ arithmetic operators:

- + addition
- - subtraction
- * multiplication
- / division
- % modulus operator
- +, -, *, and / can be used with integral and floating-point data types
- Operators can be **unary** or **binary**

Example 2-3

Arithmetic Expression	Results	Description
$5 / 2$	2	In the division $5 / 2$, the quotient is 2 and the remainder is 1. Therefore, $5 / 2$ with the integral operands evaluates to the quotient, which is 2.
$14 / 7$	2	In the division $14 / 7$, the quotient is 2 and remainder is 0. Therefore, $14 / 7$ with integral operands evaluates to 2
$34 \% 5$	4	In the division $34 / 5$, the quotient is 6 and the remainder is 4. Therefore, $34 \% 5$ with the integral operands evaluates to the remainder, which is 4.
$4 \% 6$	4	In the division $4 / 6$, the quotient is 0 and the remainder is 4. Therefore, $4 \% 6$ with the integral operands evaluates to the remainder, which is 4.

Example

$$5.0 + 3.0$$

$$3.0 + 9.4$$

$$16.3 - 5.2$$

$$4.2 * 2.5$$

$$5.0 / 2.0$$

$$34.5 / 6.0$$

$$34.5 / 6.5$$

C++ Math Operator - PEMDAS Rule

- `*` : Multiplication
- `/` : Division
 - Integer division truncates remainder
 - `7 / 5` evaluates to 1
- `%` : Modulus operator returns remainder
 - `7 % 5` evaluates to 2
- `+` : Addition
- `-` : Subtraction

Operator	Operation (s)	Order of evaluation (precedence)
<code>()</code>	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.
<code>*</code> , <code>/</code> or <code>%</code>	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
<code>+</code> or <code>-</code>	Addition, Subtraction	Evaluated last. If there are several, they are evaluated left to right

Order of Precedence

- All operations inside of $()$ are evaluated first
- $*$, $/$, and $\%$ are at the same level of precedence and are evaluated next
- $+$ and $-$ have the same level of precedence and are evaluated last
- When operators are on the same level
 - Performed from left to right (**associativity**)
- $3 * 7 - 6 + 2 * 5 / 4 + 6$ means
$$(((3 * 7) - 6) + ((2 * 5) / 4)) + 6$$

Example

$$\begin{aligned} &(((3 * 7) - 6) + ((2 * 5) / 4)) + 6 \\ = &((21 - 6) + (10 / 4)) + 6 && \text{(Evaluate *)} \\ = &((21 - 6) + 2) + 6 && \text{(Evaluate /. Note that this is an integer division.)} \\ = &(15 + 2) + 6 && \text{(Evaluate -)} \\ = &17 + 6 && \text{(Evaluate first +)} \\ = &23 && \text{(Evaluate +)} \end{aligned}$$

Example

In the expression:

$$3 + 4 * 5$$

$*$ is evaluated before $+$. Therefore, the result of this expression is 23. On the other hand, in the expression:

$$(3 + 4) * 5$$

$+$ is evaluated before $*$ and the result of this expression is 35.

Expressions

- If all operands are integers
 - Expression is called an integral expression
 - Yields an integral result
 - Example: $2 + 3 * 5$
- If all operands are floating-point
 - Expression is called a floating-point expression
 - Yields a floating-point result
 - Example: $12.8 * 17.5 - 34.50$

Examples

- Consider the following C++ integral expressions:
 - $2 + 3 * 5$
 - $3 + x - y / 7$
 - $x + 2 * (y - z) + 18$
- In these expressions, x, y, and z represent variables of the integer type; that is, they can hold integer values

Examples

- Consider the following C++ floating-point expressions:
 - $12.8 * 17.5 - 34.50$
 - $x * 10.5 + y - 16.2$
- Here, x and y represent variables of the floating-point type; that is, they can hold floating-point values

Mixed Expressions

- Mixed expression:
 - Has operands of different data types
 - Contains integers and floating-point
- Examples of mixed expressions:
 - $2 + 3.5$
 - $6 / 4 + 3.9$
 - $5.4 * 2 - 13.6 + 18 / 2$

Mixed Expressions (cont'd.)

- Evaluation rules:

Rule # 1

- a. If operator has same types of operands
 - Evaluated according to the type of the operands
- b. If operator has both types of operands
 - Integer is changed to floating-point with decimal part “0”
 - Operator is evaluated
 - Result is floating-point

Rule # 2

- Entire expression is evaluated according to precedence rules

Examples

Mixed Expression

$3 / 2 + 5.5$

$15.6 / 2 + 5$

$4 + 5 / 2.0$

$4 * 3 + 7 / 5 - 25.5$

Exercise

- Solve this expression on notebook
- $10 * 5 + 100 / 10 - 5 + 7 \% 2$



Example

- $3 / 2 + 5.5$
- $15.6 / 2 + 5$
- $4 + 5 / 2.0$
- $4 * 3 + 7 / 5 - 25.5$

Type Conversion (Casting)

- **Implicit type coercion:** When value of one type is automatically changed to another type
 - $2 + 3.4 = 2.0 + 3.4 = 5.4$
 - `int num1;`
 - `float num2 = 12.45;`
 - `num1 = num2;`
 - `num1 = ?`
- **cast operator:** provides explicit type conversion

`static_cast<dataTypeName>(expression)`

Type Conversion (cont'd.)

Expression

Evaluates to

```
static_cast<int>(7.9)
static_cast<int>(3.3)
static_cast<double>(25)
static_cast<double>(5 + 3)
static_cast<double>(15) / 2
```

```
static_cast<double>(15 / 2)
```

```
static_cast<int>(7.8 +
static_cast<double>(15) / 2)
```

```
static_cast<int>(7.8 +
static_cast<double>(15 / 2))
```

Type Conversion (cont'd.)

- You can also use cast operators to explicitly convert
- `char` data values into `int` data values
- `int` data values into `char` data values
- To convert `char` data values into `int` data values, you use a collating sequence
- For example, in the ASCII character set
- `static_cast<int>('A')` is 65 and `static_cast<int>('8')` is 56
- Similarly, `static_cast<char>(65)` is 'A' and `static_cast<char>(56)` is '8'

Questions

