COMP0204: Introduction to Programming for Robotics and AI

Programming Fundamentals

Course lead: Dr Sophia Bano

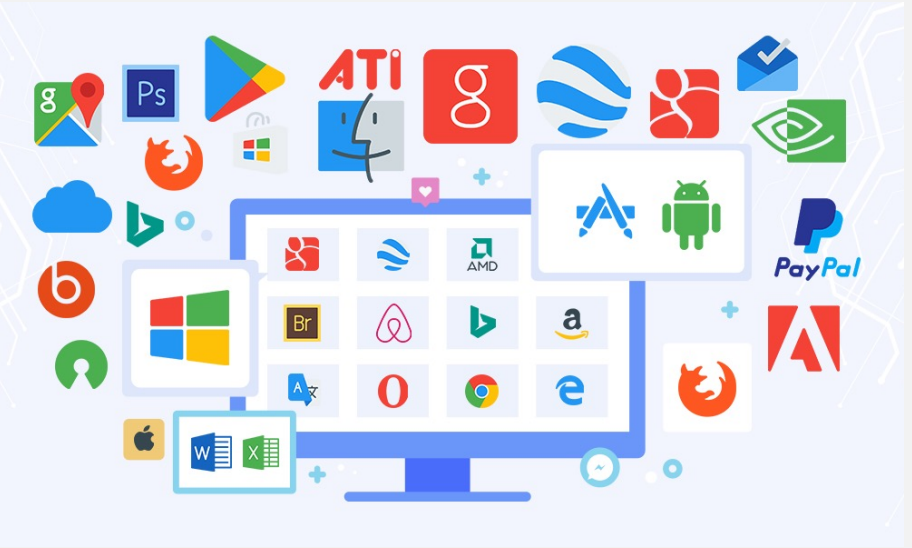
MEng Robotics and AI

UCL Computer Science

Today's lecture

- Overview of C programming
- Setting up the C development environment
- Basic C syntax and structure
- Variables and data types
- Operations

What is a Program?

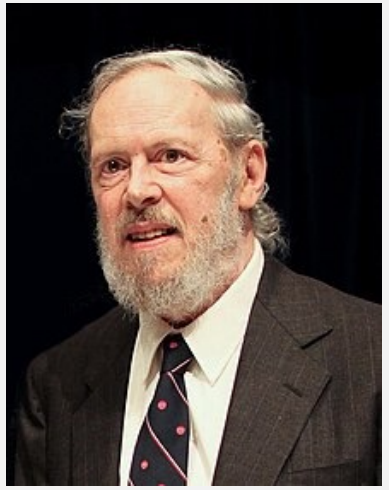


- Sequence of **instructions** and **algorithms**
- Written in a **programming language**
- Uses **algorithmic logic** (set of well-defined rules)
- Performs **data processing** (manipulation, interact with users/devices)
- Enable a computer to perform **specific tasks** or **operations**

What is C programming?

```c
#include <stdio.h>

void main()
{
    printf("hello world!\n");
    return;
}
```

- Latin of programming languages
- Created by Dennis M. Ritche
- 1972 – AT&T Bell Labs
- Remains one of the top computer programming language
  - Extends to newer system architectures
  - Efficiency/performance
  - Low-level access
- Knowing C makes learning other languages easy



Dennis M. Ritche

What is needed to write C programs?
Integrated Development Environment (IDE)

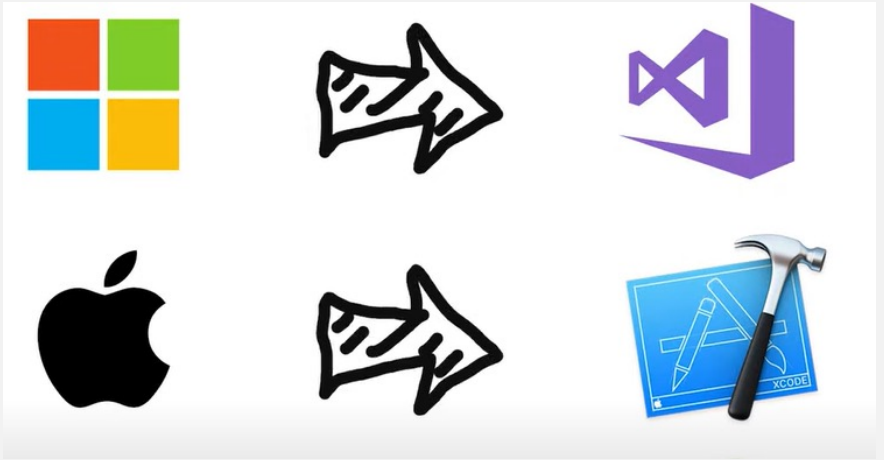
An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development.

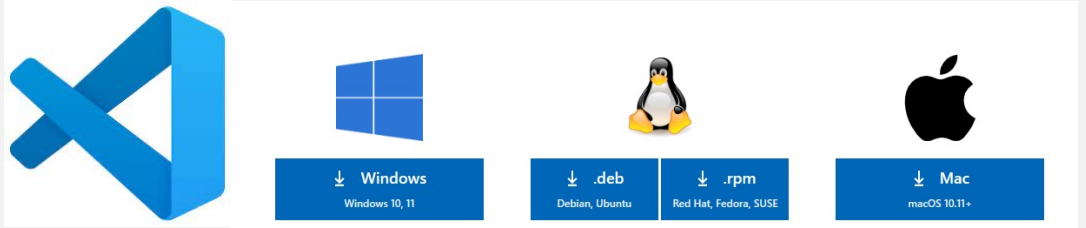
An IDE combines editor, compiler and linker and includes debugging and other tools.

Many IDE are available:

- Microsoft launched visual studio
- Apple has Xcode
- Google has Android Studio
- Eclipse from Eclipse

…

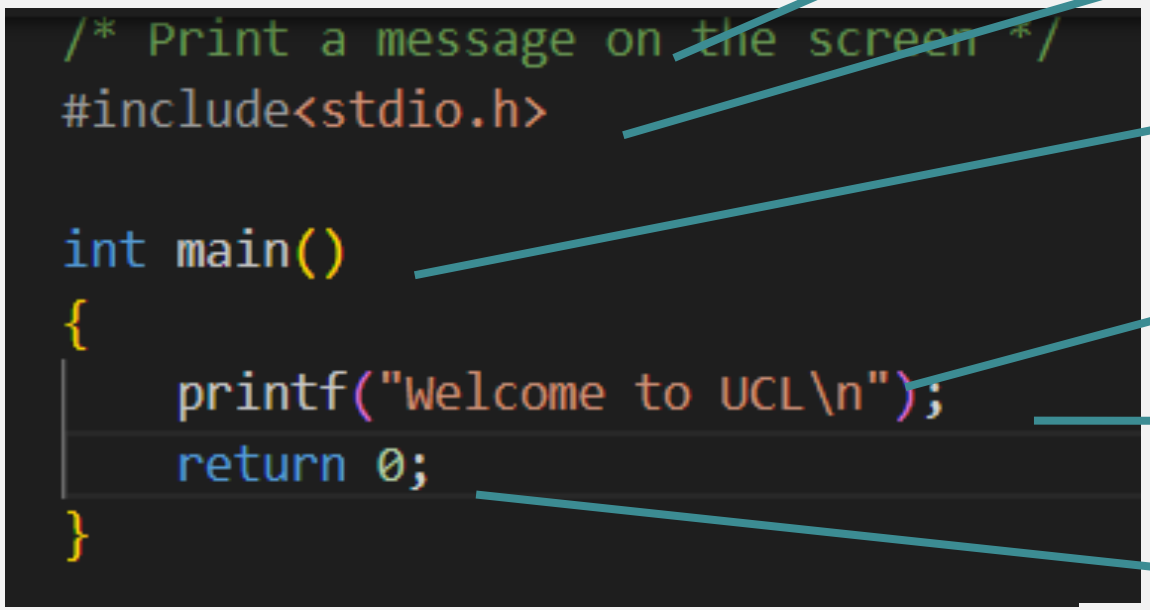




Recommended IDE**: Visual Studio Code or Code::Blocks**

Write your first C program

- Save with .c extension



Comment – to ensure your code is readable

Preprocessor (header files)– basic I/O facilities

define a function called main
- entry point for C program

\n – C notation for newline character

Statements in C ends with ';'

Return keyword passing an integer value to the operating system

- Indentation makes the code easier to read (formatted automatically in IDEs

Compile and link (build)

- Compiler translates code into object (.o) code
- Linker combines object code with C libraries to make a program

VS Code: ‘Run code’ does compile, link and run in one click

Declaring variables

- Must be declared before use

```
char c; // character data type
int i; // integer data type
float f; // floating point data type
```

More on data types later…

Initializing variables

- Variables are containers for storing data values, like numbers and characters

```
int i; // integer data type
int x, y, z = 5; // multiple integer data types
float f = 3.14; // floating point data type


i = 12; // initialize i to 12
```

Strings - input and output

String input in C:

- scanf()
  - no limitation on size
  - only reads until encounter space
- gets()
  - no limitation on size
- fgets()
  - Limit size to declared one

Covered in yesterday's lab!

String output in C:

- printf() – prints in one line
- puts() – pass control to next line after printing

```c
#include<stdio.h>
int main()
{

    // array to store string taken as input
    char color[20];

    // take user input
    printf("Enter a color: ");
    scanf("%s", color);

    //print the input value
    printf("You entered: %s\n", color);

    return 0;
}
```

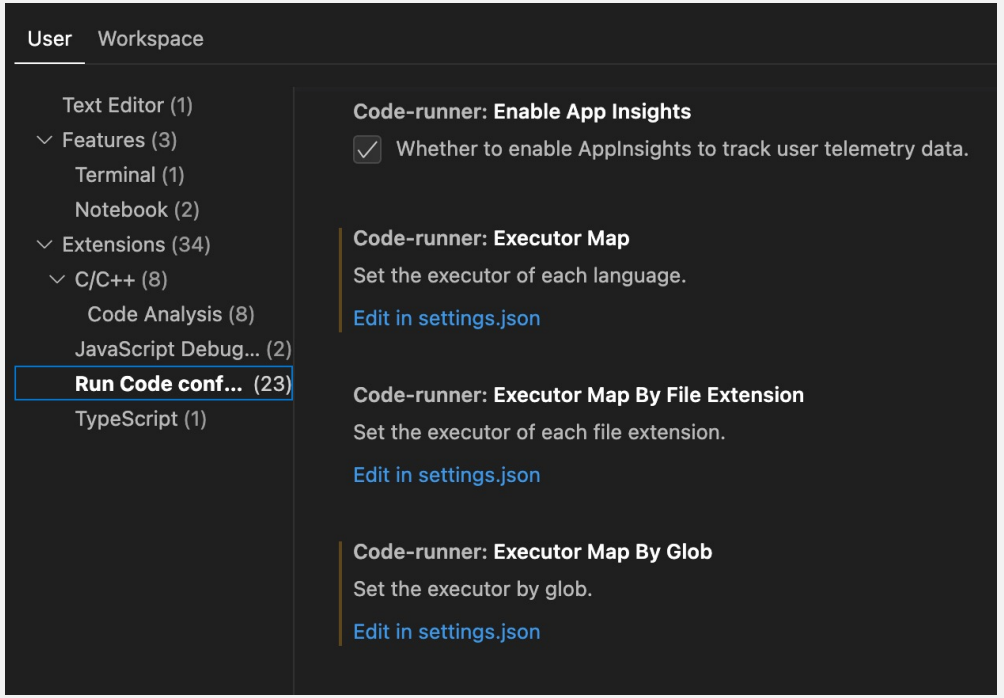
How to use getch() in MacOS?

- Install ncurses

```
brew install ncurses
```

Go in VS code settings → Under user
→ C/C++ → run code conf
- Look for Code-runner: Executor map
-Click on ‘edit settings.json’

Add –lncurses as highlighted




```
{
    "workbench.colorTheme": "Default Dark Modern",
    "code-runner.runInTerminal": true,
    "debug.onTaskErrors": "debugAnyway",
    "code-runner.executorMap": {

        "javascript": "node",
        "java": "cd $dir && javac $fileName && java $fileNameWithoutExt",
        "c": "cd $dir && gcc $fileName -o $fileNameWithoutExt -lncurses && $dir$fileNameWithoutExt",
        "zig": "zig run",
        "cpp": "cd $dir && g++ $fileName -o $fileNameWithoutExt && $dir$fileNameWithoutExt",
        "objective-c": "cd $dir && gcc -framework Cocoa $fileName -o $fileNameWithoutExt && $dir$fileNameWithoutExt",
```

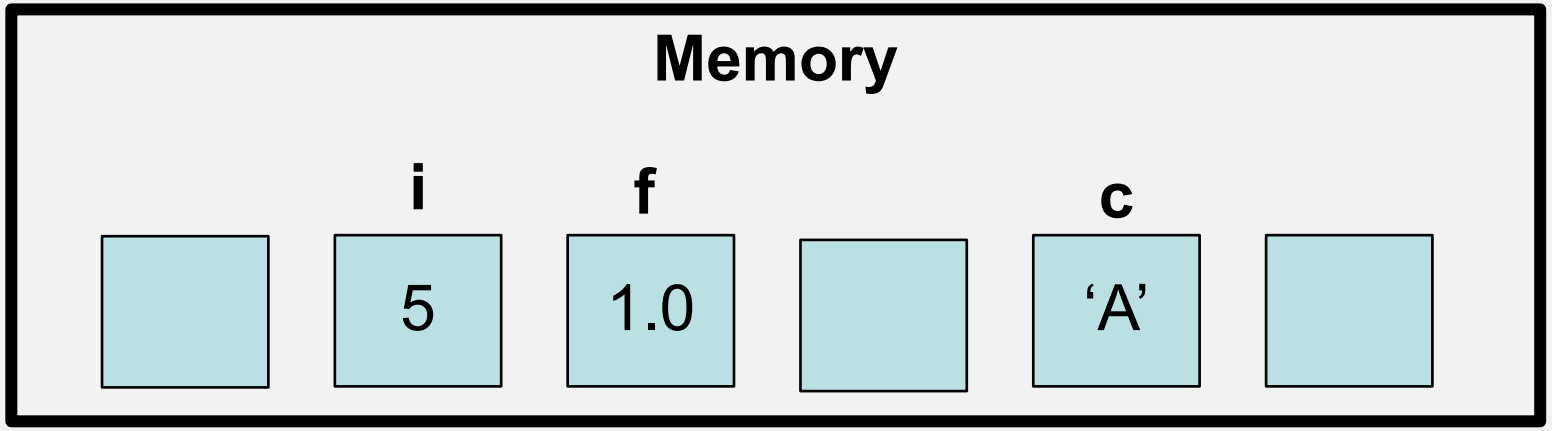
How to use getch() in MacOS?

```c
// Write a code that takes your grade as input and displays your grade.

#include <stdio.h>
#include <ncurses.h>

int main()
{
    int grade;

    /* Curses Initialisations */
    initscr();
    raw();
    keypad(stdscr, TRUE);
    noecho();

    printw("Enter your grade:\n");
    grade = getch();

    printw("\nYour grade is %c", grade);
    getch();

    return 0;
}
```

Variables and Data types

Variables

- Variable is the name of a memory location which stores some data.



Variables names

- Case sensitive (int x; int X declares two different variables)
- Include letters (upper and lower case), digits, and underscore (_)
- Can start with a letter (preferred) or underscore (_)
- Cannot contain spaces and cannot start with digits
- C 32 keywords (e.g., for,while etc.) cannot be used as variable names

Variables names

3students	studentAge	bool	num_of_apples
counter123	MAX#VAL	UserInput	Firstname
float	isInitialised	user input	for
One&two	123value	myArraySize	MAX_VALUE
_value	My-variable	firstName	total@amount

Valid	Invalid

Variables names

3students	studentAge	bool	num_of_apples
counter123	MAX#VAL	UserInput	Firstname
float	isInitialised	user input	for
one&two	123value	myArraySize	MAX_VALUE
_value	My-variable	firstName	total@amount

Valid	Invalid

Data types – Integer types

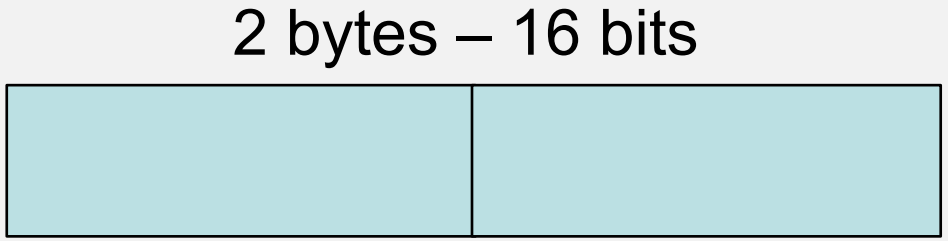
Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

To determine the exact size of a type or a variable, use sizeof() operator

```
printf("%d",sizeof(int));
```

Data types – Integer types

Short



- n bits → 2^n values (binary representation)
- 16 bits → 2^{16} = 65,536 values
- Value range: -32,768 to 32,767 (-2^{15} to $2^{15}-1$)

More on information representation in the next lecture

- Value range of a 4 bytes integer?

Data types – Integer types - Example

```c
#include <stdio.h>

int main() {
    // Unsigned integer variable
    unsigned int positiveNumber = 42;

    // Short integer variable
    short int smallNumber = -32768;
    // Short integers can hold small values, both positive and negative

    // Long integer variable
    long int largeNumber = 1234567890;
    // Long integers can hold larger values

    // Output using printf
    printf("Unsigned Integer: %d\n", positiveNumber);
    printf("Short Integer: %d\n", smallNumber);
    printf("Long Integer: %d\n", largeNumber);

    return 0;
}
```

Data types – Floating-Point Types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Data types – The void Type

Sr.No.	Types & Description
1	**Function returns as void** There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

Variable declaration

Syntax

data_type variable_name;

In C, variables must be declared before expression

```c
int age;               // Declaration of an integer variable
int num_students = 25; // Declaration and initialization of an integer variable
int x = 10, y = 20;    // Multiple declarations and initializations

float temperature;              // Declaration of a floating-point variable
float pi = 3.14;              // Declaration and initialization of a floating-point variable

char grade;             // Declaration of a character variable
char initial = 'A';     // Declaration and initialization of a character variable

int numbers[5];                 // Declaration of an integer array
int scores[] = {95, 88, 72};     // Declaration and initialization of an integer array
```

Operators

Arithmetic operations

Used to perform various mathematical operations on numeric values

- Addition (‘+’)
- Subtraction (‘+’)
- Multiplication (‘*’)
- Division (‘/’)
- Modulus (‘%’)

```c
#include <stdio.h>

int main()
{
    float x = 5;
    float y = 2;

    printf("%f\n", x+y);
    printf("%f\n", x-y);
    printf("%f\n", x*y);
    printf("%f\n", x/y);

    return 0;
}
```

Arithmetic operations

Pre-increment/pre-decrement (++x, --x)

- ++x is a short cut for x=x+1
- −−x is a short cut for x=x−1
- y=++x is a short cut for x=x+1;y=x;. x is evaluate after it is incremented
- y=−−x is a short cut for x=x−1;y=x;. x is evaluate after it is decremented

```
int a = 1;
int b = ++a;

int a = 1;
int b = a++;
int c = a;
```

Arithmetic operations

Post-increment / post-decrement (x++, x--)

- x++ is a short cut for x=x+1
- x−− is a short cut for x=x−1
- y=x++ is a short cut for y=x; x=x+1. x is evaluated before it is incremented
- y=x−− is a short cut for y=x; x=x−1. x is evaluated before it is decremented

```
int a = 1;
int b = ++a;

int a = 1;
int b = a++;
int c = a;
```

Arithmetic operations

- Assignment (‘=‘): Assigns the value on the right to the variable on the left

```
int a = 10;
int b = a + 5;   // b is assigned the value 15
```

- Compound Assignment: Combines an arithmetic operation with assignment

```
int c = 3;
c += 2;    // Equivalent to c = c + 2; c is now 5
```

Arithmetic operations

Exercise

- Given the length and width of the rectangle, write a C program to calculate and display its area.

Relational operators

Used to compare two values or expressions and evaluate whether a certain relationship between them is true or false

Operator	
Equal to (‘==‘)	Checks if two values are equal
Not equal to (‘!=‘)	Checks if two values are not equal
Greater than (‘>’)	Checks if the left operand is greater than the right operand
Less than (‘<‘)	Checks if the left operand is less than the right operand
Greater than or equal to (‘>=‘)	Checks if the left operand is greater than or equal to the right operand
Less than or equal to (‘<=‘)	Checks if the left operand is less than or equal to the right operand

```
int x = 5;
int y = 5;
if (x == y) {
    // This condition is true
}
```

Commonly used in conditional statements (e.g., ‘if’, ‘else if’, ‘while’, ‘for’) to control the flow of a program based on specific conditions.

Conditional Expression (Syntax)

```
if (condition)
x=<expression_if_true>;
else
x=<expression_if_false>;
```

```
condition ? expression_if_true :
expression_if_false;
```

C provides syntactic sugar to express the same using the

ternary operator ’?:’

The ternary operator makes the code shorter and

easier to understand (syntactic sugar).

Relational operators

Exercise

Write a C program that compares two numbers, num1 and num2, and displays whether num1 is greater than and equal to num2.

Logical operators

- Used to combine and manipulate Boolean values (true or false) to make decisions and control the flow of a program

1. && (logical AND)
2. || (logical OR)
3. ! (logical NOT)

- Often used in combination with relational operators to create complex conditions in conditional statements (‘if’, ‘else if’, ‘while’, ‘for’) and loops to control the program's behavior based on multiple conditions

Logical operators

Exercise

- You are asked to check driving license record of users in the UK. Users can drive independently if they meet both of the following conditions:
    1. Age 18 and above
    2. Has full license
- Write a C program that checks if a user can drive or not

Bitwise operators

- Used for performing operations on individual bits of integers (usually ‘int’ and ‘char’ data types)
- Allow to manipulate the binary representation of integers

Operator	
Bitwise AND (&)	If both bits are 1, the result bit is 1; otherwise, it's 0
Bitwise OR (|)	If at least one of the bits is 1, the result bit is 1; otherwise, it's 0
Bitwise XOR (^)	If the bits are different (one is 0 and the other is 1), the result bit is 1; otherwise, it's 0
Bitwise NOT (~):	1s become 0s, and 0s become 1s
Bitwise Left Shift (<<)	Shifts the bits of an integer to the left by a specified number of positions
Bitwise Right Shift (>>):	Shifts the bits of an integer to the right by a specified number of position

Bitwise operators

- Examples

```c
int a = 12;
int b = 9;
int result = a & b;
printf("a & b = %d\n", result);

int x = 5;
int y = 3;
int result = x | y;
printf("x | y = %d\n", result);


int m = 10;
int n = 6;
int result = m ^ n;
printf("m ^ n = %d\n", result);


int value = 7;
int result = ~value;
printf("~value = %d\n", result);


int number = 8;
int shifted = number << 2;
printf("number << 2 = %d\n", shifted)

int num = 16;
int shifted = num >> 2;
printf("num >> 2 = %d\n", shifted);
```

Conditional Expression

Example

```c
#include <stdio.h>

int main() {
    int x = 10;
    int y = 20;
    int max;

    // standard syntax for condition expression
    if (x > y) {
        max = x;
    } else {
        max = y;
    }

    // syntatic sugar for condition expression
    //max = (x > y) ? x : y;

    printf("The maximum is: %d\n", max);

    return 0;
}
```

Type conversions

- **Implicit Type Conversion:** When variables are promoted to higher precision, data is preserved. This is automatically done by the compiler for mixed data type expressions.

```
int x = 5;
float y = 3.14;

float result = x + y;  // Implicit conversion of 'x' to float
```

- **Explicit Type Conversion (Type Casting):** This is performed explicitly by the programmer using casting operators.

```
int num1 = 10;
double num2 = (double)num1;  // Explicit cast from int to double
```

Precedence and order of evaluation

B	**B**rackets first
O	**O**rders (i.e. Powers and Square Roots, etc.)
DM	**D**ivision and **M**ultiplication (left-to-right)
AS	**A**ddition and **S**ubtraction (left-to-right)

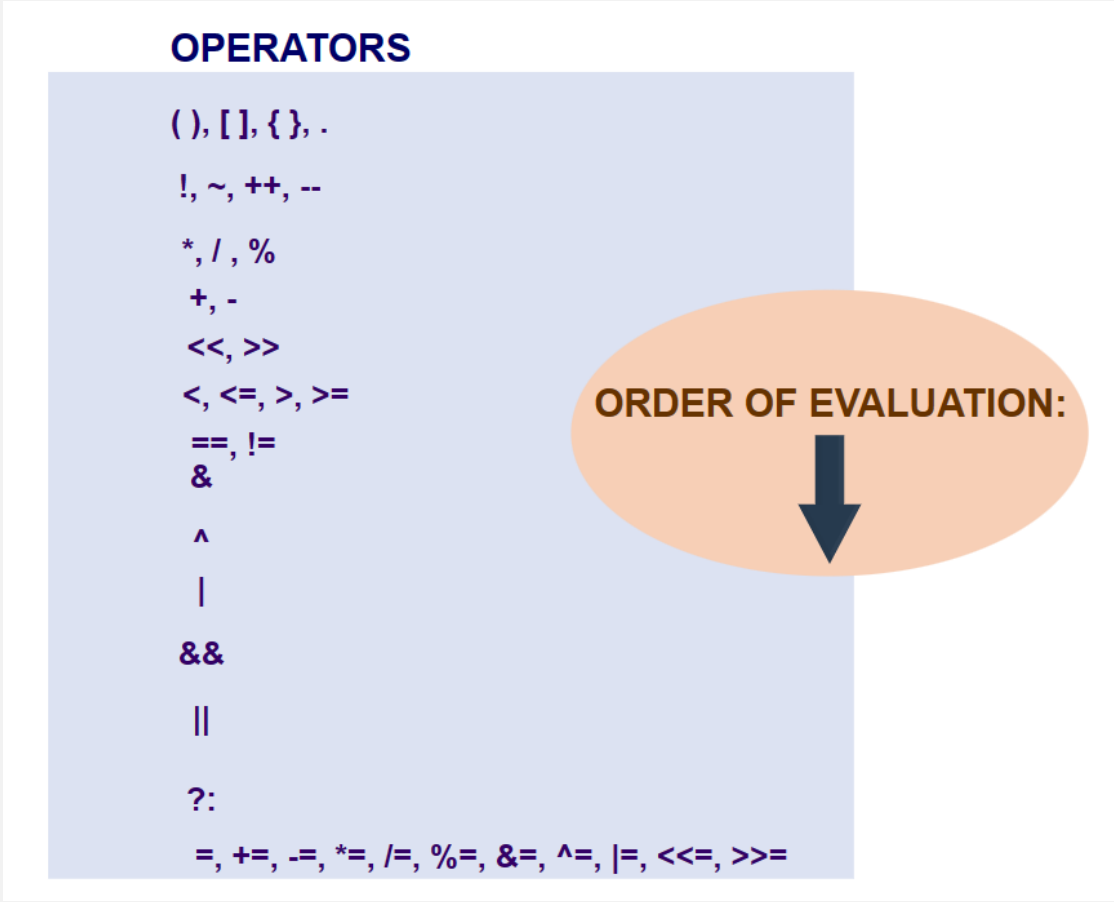
Order of operations:

Operator	Evaluation direction
*,/,%	left-to-right
+,-	left-to-right
=,+=,-=,*=,/=,%=	right-to-left

Use () to avoid ambiguities or side effects associated with precendence of operators

Examples:

```
y=x*3+2; //same as y=(x*3)+2

x!=0 && y==0 //same as (x!=0) && (y==0)

d = c>='0'&& c<='9' //same as d=(c>='0') && (c<='9')
```



Summary – What did we cover?

- **Overview**: Introduction to C programming
- **Environment Setup**: Configuring your development environment for C programming
- **Syntax & Structure**: Understanding the basic syntax, statements in C
- **Variables & Data Types**: Exploring how to declare and use variables of different data types
- **Operations**: Introduction to fundamental operations such as arithmetic, relational, logical, and bitwise operations

Happy Coding!: Practice, collaborate, and embrace the problem-solving journey in programming.