week 1_1:

programming basics

rajin teaches programming

– OVERVIEW:

This week, I'll lay the foundation for understanding **programming** concepts and delve into the **basics** of the C programming language. By the end of this week, you'll have a solid understanding of how programming **works**, the **essentials** of C programming, and the significance of C in the world of **software development.**

UNDERSTANDING PROGRAMMING:

– WHAT IS PROGRAMMING?

Programming is the **process** of **instructing** a **computer** to perform **specific** tasks using a set of **predefined instructions.** It involves designing **algorithms**, writing **code**, **testing**, **debugging**, and **maintaining** programs.

– WHY LEARN PROGRAMMING?

Programming enhances **problem-solving** skills, **logical** thinking, and **creativity.** It opens doors to various **career** opportunities in **software development**, **data science**, **artificial intelligence**, and more.

INTRODUCTION TO C:

– WHAT IS C?

C is a powerful and widely-used programming language developed in the early **1970s** by **Dennis Ritchie** at **Bell Labs.** It is known for its **efficiency**, **portability**, and **flexibility**, making it suitable for **system** programming and developing **operating systems.**

– WHY LEARN C?

C provides a strong foundation for understanding programming **principles** and **concepts.** It offers **low-level** access to the system's **memory** and **hardware**, allowing for **efficient** and **high-performance** code. Many **modern** programming languages are **influenced** by C, making it a **valuable** language to learn.

```c
#include<stdio.h>

int main() {

    printf("Hello world!\n");

    return 0;
}
```

preprocessor directive/header file

main function

code (print statement)

return statement

a typical C program.

– MAIN FUNCTION AND PREPROCESSOR DIRECTIVES:

Every C program **starts execution** from the `**main()**` function, which serves as the **entry point** of the program.
Preprocessor directives, such as `**#include**`, are used to include **header files** and perform text substitution **before compilation.** **Header files** store **library functions** that we need to create basic programs. As such, "**stdio.h**", meaning the **standard input output** header file, stores the **definitions** of basic functions for input and output, such as the **printf()** function and the **scanf()** function.

– THE printf() FUNCTION:

printf() is a **function** in C used to **print** formatted **output** to the **standard output** (usually the **console**). It accepts **format specifiers** and **arguments** for output formatting. Additionally, every statement in C ends with a semicolon "**;**" (Except preprocessors).

– OUTPUTTING STRINGS:

Strings in C are **enclosed** in **double quotes** " ".
For example: `printf("Hello, World!\n");`
We can also use the `**%s**` **format specifier** to output strings, and I'll explain how format specifiers work in class.

– OUTPUTTING NUMBERS (AND CHARACTERS):

For printing **numbers**, we use format specifiers such as `%d` for **integers**, `%f` (or `.intf`) for **floats**, and `%c` for **characters** (In printf(“”)). We will get into the different data types soon.

Developer - output_num.c

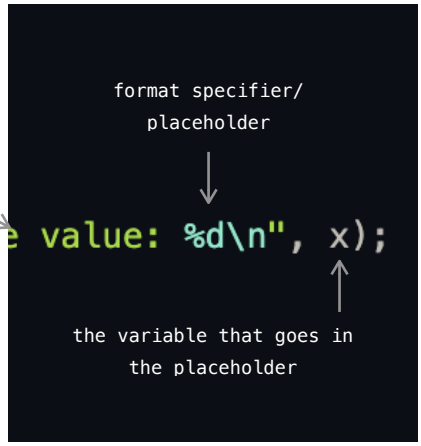
```c
#include<stdio.h>

int main() {

    int x;
    x = 7;

    printf("The variable x stores the value: %d\n", x);

    return 0;
}
```



output

```
The variable x stores the value: 7
```

– ESCAPE SEQUENCES:

These are **special characters** used to represent **non-printable** characters and **formatting** options. Common escape sequences include `\n` for **newline**, `\t` for **tab**, `\"` for **double quote**, and `\\` for **backslash**.

DATA TYPES AND SIZES (IN C):

C supports **various** data types such as **`int`**, **`float`**, **`char`**, **`double`**, etc. Each type has a **specific size** in memory.

int	stores **integers** (**4 bytes**)
float	stores numbers with **floating point precision** (**4 bytes**)
double	stores **larger** numbers (**8 bytes**)
char	stores a **single character** (“a”, or “@“, or “%”) (**1 byte**)
bool	stores a **boolean** value (**either TRUE/1 or 0/FALSE**) (**4 bytes**)

MATHEMATICAL OPERATORS (AND PRECEDENCE):

C supports **arithmetic operators** such as **addition** `+`, **subtraction** `-`, **multiplication** `*`, **division** `/`, and **modulo** `%` (**remainder**). The **order of precedence** of operators determines the **sequence** of evaluation:

1. **Parentheses `()`**
2. **Multiplication `*`, Division `/`, Modulo `%`**
3. **Addition `+`, Subtraction `-`**

Parentheses can be used to **alter** the **default precedence** and **enforce** a **specific order** of evaluation.

STATICALLY TYPED VS DYNAMICALLY TYPED LANGUAGES:

- STATICALLY TYPED LANGUAGE:

In **statically typed** languages like C, **data types** are determined at **compile-time. Variables** must be **declared** with their **data types before** they can be used. This is commonly known as **initialization.** An example with a separate explanation will be shown afterwards. This process usually provides **type safety** and **early detection** of **errors.**

- DYNAMICALLY TYPED LANGUAGE:

In **dynamically typed** languages like Python, data types are determined at **runtime.** Variables can **change** their **data types** during **execution.** Offers **flexibility** but may lead to **runtime errors** if **type mismatches** occur.

- INITIALIZATION OF VARIABLES:

Initialization refers to **assigning** an **initial** value to a **variable** when it is **declared.** In C, variables should be initialized to **avoid** accessing **undefined** values. Using the previous example, the initialization step has been outlined:

```c
#include<stdio.h>

int main() {

    int x;
    x = 7;

    printf("The variable x stores the value: %d\n", x);

    return 0;
}
```

initialization

the same example, showing specific lines for initialization

The **first** line is the **declaration** of the variable, while the **second** line is the **initialization.** This same initialization could be achieved in **one line**, by writing "`int x = 7;`", and this is **more useful** when using a **single variable.** However, when creating **large programs** or software which may need a plethora of variables, it is **more useful** to first **declare** the variables in **one line**, then **initialize** it in a **separate one.**

IMPORTANCE OF PRACTICE:

Programming **skills improve** with **practice** and **hands-on experience. Experiment** with **code, solve problems,** and explore **different programming constructs** to **reinforce** learning.

SOME INTERESTING STATISTICS:

– POPULARITY OF C:

According to the TIOBE **Index,** C **consistently** ranks among the **top programming languages** in terms of **popularity** and **usage.** C is widely used in **industries** such as **system programming, embedded systems, game development,** and more.

CONCLUSION:

And with that, you've completed the basics of programming with C. Keep practicing the basics, and we'll delve into more interesting concepts next class.

next class 1_2:

arithmetic operations and conditionals

rajin teaches programming

reach out for classes at: rajin.khan2001@gmail.com