week 1_2:

math, user inputs and constants

rajin teaches programming

– OVERVIEW:

This week, we'll build upon the foundation laid in the first few classes, and explore more the final concepts in the basics of C programming. By the end of this class, you'll have a solid understanding of **C syntax**, handling **user input**, performing **math operations**, **typecasting**, and using **constants** in your programs.

C SYNTAX:

– UNDERSTANDING C SYNTAX:

C programs are composed of **functions**, **variables**, **statements**, and **expressions**.
Statements are terminated by a **semicolon** `;`.
Blocks of code are enclosed within **curly braces** `{ }`.

USER INPUT IN C:

– THE scanf() FUNCTION:

The **scanf() function** is used to read **input** from the **standard input** (usually the **keyboard**). It accepts **format specifiers** to specify the **type** of input to be **read**:

Developer - input.c

```c
#include<stdio.h>

int main() {

    int num;

    printf("Enter a number: ");
    scanf("%d", &num);

    printf("The number you have entered is: %d", num);

    return 0;
}
```

an example of taking an input from the user.

In **scanf("%d", &num);**, the `&` symbol is the **"address of" operator**, which returns the **memory address** of the **variable** num. This allows **scanf()** to **store** the **input** value **directly** at that **memory location**.

– UNDERSTANDING INPUT BUFFER AND fflush():

Input buffer is a **temporary storage** area for **input** data.
After using **`scanf()`**, it's important to **clear** the input buffer using **`fflush(stdin);`** to **prevent unexpected behavior:**

Developer - buffer.c

```c
#include<stdio.h>

int main() {

    int char1, char2;

    printf("Enter two characters: ");
    scanf("%c", &char1);

    fflush(stdin);

    scanf("%c", &char2);

    printf("The two characters you have entered are: %d, %d", char1, char2);

    return 0;
}
```

an example of clearing the input buffer.

– INPUT FORMAT SPECIFIERS:

Use **format specifiers** **`%d`** for **integers**, **`%f`** for **floats**, **`%c`** for **characters**, **`%s`** for **strings**, etc.

MATH OPERATIONS:

– ARITHMETIC OPERATIONS:

C supports **standard arithmetic** operations: **addition** **`+`**, subtraction **`-`**, **multiplication** **`*`**, **division** **`/`**, and **modulo** **`%`** (returns the **remainder**, useful for checking **factors**).

Developer - math.c

```c
#include<stdio.h>

int main() {

    int num1, num2;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    int sum = num1 + num2;
    int diff = num1 - num2;
    int prod = num1 * num2;
    float div1 = (float)num1 / num2;
    float div2 = (float)num2 / num1;

    printf("The sum of the two numbers are: %d\n", sum);
    printf("The difference between the two numbers are: %d\n", diff);
    printf("The product of the two numbers are: %d\n", prod);
    printf("%d/%d: %.2f\n", num1, num2, div1);
    printf("%d/%d: %.2f\n", num2, num1, div2);

    return 0;
}
```

an example of basic math operations in C.

you may additionally use the <math.h> library for advanced math functions.

– MATH OPERATIONS WITH float VS int:

Math operations with **float** produce **floating-point** results, while operations with **int** produce **integer** results. Be mindful of **data types** when **performing** math operations to avoid **unexpected results.**

TYPECASTING:

– WHAT IS TYPECASTING?

Typecasting is the process of **converting** a **value** from **one data type** to **another.** It can be done **implicitly** by the compiler or **explicitly** by the programmer using casting operators.
Example: `floatNum = (float) intNum;`
explicitly converts an integer `**intNum**` to a **float.**
Typecasting is **useful** when you need to perform operations involving **different data types** or when you want to ensure **data integrity.**

Developer - typecasting.c

```c
#include<stdio.h>

int main() {

    int num;

    printf("Enter an integer: ");
    scanf("%d", &num);

    printf("The integer is: %d\n", num);
    printf("The float of that integer is: %f\n", (float)num);

    return 0;
}
```

```
Enter an integer: 7
The integer is: 7
The float of that integer is: 7.000000
```

an example of typecasting

CONSTANTS:

– UNDERSTANDING CONSTANTS:

Constants are **values** that do **not change** during program **execution.** They are **defined** using the `const` **keyword** and can be of **any** data type**.** Example: `const int MAX_VALUE = 100;` defines a constant integer with a value of `100`**.** Constants are helpful for making your code more **readable, maintainable,** and **self-documenting.** They also help **prevent accidental changes** to **important** values in your program**.**

SCENARIOS WHERE TYPECASTING AND CONSTANTS MAY BE USEFUL:

– TYPECASTING:

Useful when performing **arithmetic operations** involving **different data types,** such as **mixing** integers and floats**.**
Also used when **passing arguments** of different data types to **functions** that expect specific types (to be discussed **later**)**.**

– CONSTANTS:

Useful when **defining values** that should **not** be changed during program execution**,** such as **mathematical constants** (`PI`)**.**
Also used when defining **configuration values** or **parameters** that may need to be **adjusted** in the **future** but should remain constant within a **single execution** of the program**.**

CONCLUSION

Congratulations on completing series 1 of **Introduction** to Programming with C**.** You've explored today the **final** of basic concepts in C programming**,** including **syntax,** handling user **input,** performing **math** operations**, typecasting,** and **constants.** These concepts form the **building blocks** for writing more **complex** and **sophisticated** programs in C**.**

Keep **practicing** and **experimenting** with code to **reinforce** your understanding**.** If you have any **questions** or need further **clarification,** don't hesitate to ask**.**

next class 2_1:

conditional statements

rajin teaches programming

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