Introduction to C

1 Introduction

C is a general purpose, imperative programming language. Many programming languages, including Python, borrowed the control structures and features of C.

Try this!
Type the code below in a text editor and save it in your computer's `Desktop` folder.

```c
// filename: program.c
#include<stdio.h>        //standard library for input and output

int main(){              //main function
   char name[20];        //variable declarations
   int age;
   float grade;

   printf("What is your name?"); //printing of data
   scanf("%s", name);            //get input from the user

   printf("What is your age?");
   scanf("%d", &age);

   printf("Expected grade in CMSC 21?");
   scanf("%f", &grade);

   printf("Hi! My name is %s. I am %d years old.", name, age);
   printf("My expected grade in CMSC 21 is %f.\n", grade);

   return 0;
}// this is a comment
```

1.1 How to compile and run a C program?

A C program needs to be compiled first before it can be run.

1. In **compiling a C program**, we use the following command:

   ```
   gcc -o obj_filename filename.c
   ```

   where `gcc` is the command to run the compiler,
   `-o` is the option to create an executable file,
   `obj_filename` is the name of the executable file to be created, and
   `filename.c` is the filename of the C program.

2. **Running the program**, we use the command:

   ```
   ./obj_filename
   ```

   This command will tell the compiler to execute the program.

1.2 Main Parts of a C Program

`#include<stdio.h>`	This is a preprocessor directive which tells the GCC compiler to add or import some built-in functions of C.
`int main(){ ... }`	This is the main function of the program. Once the program is executed, all code placed in the main will be performed.

2 Variables and Data Types

Variables in C must be **declared first** before you can use them (unlike in Python).

Below are the four (4) **basic types** in C:

`int`	used to define integer numbers	`char`	used to define characters
`float`	used to define floating point numbers (precision is up to 6 decimal place or 4 bytes)	`double`	same as float but up to 15 decimal places in precision (8 bytes)

Examples:

```
char letter;       //a character variable with the name letter
int age;         //an integer variable with the name age
float weight;      //a float variable with the name weight
```

2.1 Arrays

Arrays are **fixed-size sequential collection of elements** of the same type. It is used to store a collection of data. This is used to store strings. (We will have a different lab session for this, so for now just keep in mind the syntax.)

Examples:

```
char name[10];     //array of characters with a size of 10
int num[5];        //array of integers with a size of 5
float grades[15]; //array of float numbers with a size of 15
```

3 Basic Functions, Operators, and Rules

3.1 Printing Data

To print in C, we use the function `printf` using the format below:

`printf(formatString[, value1, value2, ...]);`

Examples:

```
printf("This is a plain text."); //printing plain string
printf("Hello, %s!", name);      //printing values from variables
printf("Your age is %d!", age);
```

Format Identifiers
Format identifiers included in the format string tells the compiler that these values will be replaced with a value given by the succeeding arguments. Each data type in C has their own format specifiers, as listed below.

Data Type	Format Identifier	Data Type	Format Identifier
Integer	`%d` or `%i`	Character	`%c`
Double or Float	`%f`	String or Character Array	`%s`

3.2 Getting Inputs

To get input from the user, you can use the built-in `scanf` function using the format below:

`scanf(formatString[, varAddress1, varAddress2, ...]);`

Examples:

```
scanf("%s", name);       //scan string to variable: char name[50];
scanf("%d", &age);       //scan integer to variable: int age;
scanf("%f", &weight);    //scan float to variable: float weight;
scanf("%c", &letter);    //scan char to variable: char letter;
```

Notice that there is an ampersand (`&`) in front of the variable for the examples except for the first one. This will be discussed in full detail in a separate lab session. For now, just keep in mind that you need to put an ampersand in front of the variable if the variable is declared as `int`, `float`, `double`, or `char`.

3.3 Operators

Below are some of the operators that are available in C.

Arithmetic Operators		
Operator	**Usage**	**Description**
`+`	`a + b`	Adds two operands
`-`	`a - b`	Subtracts second operand from first operand
`*`	`a * b`	Multiplies the two operands
`/`	`a / b`	Divides numerator by denominator; becomes integer division if all operands are integers
`%`	`a % b`	(Modulo Operator) Returns the remainder of integer division to `a` and `b`
`++`	`a++` or `++a`	increases variable value by one
`--`	`a--` or `--a`	decreases variable value by one

Relational Operators		
Operator	**Usage**	**Description**
`==`	`a == b`	checks if `a` is equal to `b`
`!=`	`a != b`	check if `a` is not equal to `b`
`>`	`a > b`	checks if left value is greater than the right value
`<`	`a < b`	checks if left value is less than the right value
`>=`	`a >= b`	checks if left value is greater than or equal the right value
`<=`	`a <= b`	checks if left value is less than or equal the right value

Logical Operators		
Operator	**Usage**	**Description**
`&&`	`a && b`	Logical AND operator
`||`	`a || b`	Logical OR operator
`!`	`!a`	Logical NOT operator
`=`	`a = 2`	Assignment Operator

Bitwise Operators		
Operator	**Usage**	**Description**
`&`	`a & b`	Binary AND operator
`|`	`a | b`	Binary OR operator
`^`	`a ^ b`	Binary XOR operator
`~`	`~a`	Binary ONE'S COMPLEMENT operator
`<<`	`a >> n`	Binary left shift operator; shifts the bits of the number to the right by `n` bits
`>>`	`a << n`	Binary right shift operator; shifts the bits of the number to the left by `n` bits

Examples:

```
float ans1 = 12.0 / 5;  //will return 2.400000
int ans2 = 12 / 5;      //will return 2
int isExcessive = numAbsencesLab>3 || numAbsencesLec>6;
int isFailing = isExcessive || grade==5.0;
```

NOTE: There is no boolean data type in C. The value zero (0) is considered false while any non-zero value is considered true. That is why, in the example above, the result of the relational statements are stores in an `int`.

Precedence Rules

Arithmetic statements in C follows the **PEMDAS** rule.

Highest: Parentheses (`()`)
Multiplication (`*`), Integer Division (`/`), Modulo (`%`)
Lowest: Addition (`+`), Subtraction (`-`)

Other precedence rules will be discussed during other lab sessions.