Name: Nivedita Londhe
PRN: 22420003
Batch: EN-4
Practical No. 7

Title: Review of C Language

1. **Data Types**
   - **a) What is the need of data types**
   - ➔ Data types specify the type of data a variable can store, ensuring proper memory allocation and preventing incorrect operations.
   - **b) Step What is the size of character data type? (in byte)**
   - ➔ 1 Byte
   - **c) What is the range of values the character data type can hold?(mention signed and unsigned values)**
   - ➔ Signed: -128 to 127
     Unsigned: 0 to 255
   - **d) When to use signed char and when to use unsigned char?**
   - ➔ **Signed char:** When negative values are needed.
     Unsigned char: When only non-negative values (0 and positive) are needed, especially when higher positive values are required.
   - **e) What is the size of integer data type?**
   - ➔ Typically 4 bytes (platform-dependent).
   - **f) What is the range of values the integer data type can hold?(mention signed and unsigned values)**
   - ➔ **Signed:** -2,147,483,648 to 2,147,483,647
     Unsigned: 0 to 4,294,967,295
   - **g) What is the purpose of integer data type?**
   - ➔ To store whole numbers without decimals.
   - **h) What is the size of float data type?(in byte)**
   - ➔ Typically 4 bytes.
   - **i) What is the range of values the float data type can hold?(mention signed and unsigned values)**
   - ➔ **Signed:** Approx. -3.4E+38 to +3.4E+38
     Unsigned: Floats don't have an unsigned version; they always include positive and negative values.
   - **j) What is the purpose of float data type?**
   - ➔ To store fractional numbers or numbers with a decimal point.
   - **k) What is the size of long data type?(in byte)**
   - ➔ Typically 8 bytes.
   - **l) What is the range of values the long data type can hold?(mention signed and unsigned values)**
   - ➔ **Signed:** -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
     Unsigned: 0 to 18,446,744,073,709,551,615
   - **m) What is the purpose of long data type?**

➔ To store larger whole numbers that exceed the range of an int.

n) Why is it better to explicitly declare signed or unsigned?(For long data type)

➔ To avoid ambiguity and ensure you get the expected range and behavior when performing arithmetic operations.

o) What is the size of double data type?(in byte)

➔ Typically 8 bytes.

p) What is the range of values the double data type can hold?(mention signed and unsigned values)

➔ Signed: Approx. ±1.7E-308 to ±1.7E+308
Unsigned: Like float, no unsigned version exists.

q) What is the purpose of double data type?

➔ To store larger or more precise floating-point numbers than float.

r) What is the meaning of void?

➔ It represents the absence of any type or value, typically used for functions that do not return a value.

s) Can we declare variable of void type? Why?

➔ No, you cannot declare a variable of void type because void means "no type" and thus cannot hold a value.

t) What is the size of void data type?

➔ Void itself has no size since it represents "no data."

u) What is the purpose of void?

➔ Used mainly in functions to indicate that no value is returned or when working with generic pointers (e.g., void *).

2. Pointers in C:

a) What is a pointer?

➔ A pointer is a variable that stores the memory address of another variable.

b) What are the advantages of pointer?

- Direct access to memory.
- Efficient handling of arrays and strings.
- Dynamic memory allocation.
- Helps in implementing data structures like linked lists.

c) How to declare pointer? Why do we need to specify the data type while declaring it?

➔ int ***ptr;**
We specify the data type because the pointer must know the type of data it points to for proper memory dereferencing.

d) Write a program to list the use of pointer. And mention the possible outcomes.

➔ C code:

```c
#include <stdio.h>
int main() {
    int num = 10;
    int *ptr = &num;
    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", ptr);
    printf("Value at the address of num: %d\n", *ptr);
    return 0;
}
```

Possible outcomes:

- The value of num (10).
- The memory address of num.
- The value stored at that memory address (10).

e) What is void pointer?

➔ A pointer that can point to any data type, declared as void *.

f) What is null pointer?

➔ A pointer that doesn't point to any valid memory location, usually initialized to NULL.

3. Structure in C:

a) What is a structure?

➔ A structure is a user-defined data type that groups different data types into a single unit.

b) Write advantages of structures?

- Allows grouping of different data types.
- Facilitates handling of complex data structures like records.
- Provides better data organization.

c) How to declare structure? Why do we need to specify the data type while declaring it?

➔ C code:

```c
struct Employee {
    char name[50];
    int id;
    float salary;
};
```

We specify the data type for each member to inform the compiler how much memory to allocate for each.

d) Write a program to list the use of structure. And mention the possible outcomes.

➔ C code:

```c
#include <stdio.h>
struct Employee {
    char name[50];
    int id;
    float salary;
};
int main() {
    struct Employee emp = {"John", 101, 5000.50};
    printf("Name: %s\n", emp.name);
    printf("ID: %d\n", emp.id);
    printf("Salary: %.2f\n", emp.salary);
    return 0;
}
```

Possible outcomes:

Displays employee details like name, ID, and salary.

4. Writing portable C Code

a) Illustrate portability with 1 example.

➔ Portable code runs on different platforms without modification. Using standard libraries like stdio.h ensures portability.

Example:

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

b) Write the guidelines to be followed while writing portable code.

➔ Guidelines:

- Use standard libraries.
- Avoid platform-specific features.
- Use portable data types (int, char).
- Prefer higher-level abstractions.

c) Whether the RTOS code should be portable? Why?

➔ Yes, RTOS (Real-Time Operating System) code should be portable to enable the same software to run on multiple hardware platforms without modification, improving maintainability and flexibility.