**Programming for Problem Solving**

**WEEK-3**

**LONG DESCRIPTIVE QUESTION**

**1, Explain the storage classes in C with an appropriate example**

Storage classes in C define how and where variables are stored in memory, their lifetime, and their scope. There are four main types of storage in C: auto, static, register, and extern. Let's explore each of them with appropriate examples

**Auto Storage class:**

* Variables declared with the auto storage class are local to the block or function where they are defined.
* They have automatic storage duration, which means they are created when the block or function is entered and destroyed when it exits.
* auto is the default storage class for local variables, so it is rarely explicitly used in code.

Example:

#include <stdio.h>

int main() {

auto int a = 10; // 'a' is an auto variable

printf("a = %d\n", a);

return 0;

}

**Static Storage Class:**

* Variables declared as static have a longer lifetime than auto variables.
* They are initialized only once and retain their values between function calls.
* static variables have block scope but remain in memory throughout the program's execution.

Example:

#include <stdio.h>

void count() {

static int counter = 0; // 'counter' is a static variable

counter++;

printf("Counter = %d\n", counter);

}

int main() {

count(); // Call count() multiple times

count();

count();

return 0;

}

**Register Storage Class:**

* Variables declared as register are typically stored in CPU registers for faster access.
* The use of register is a request to the compiler, and the compiler may choose to ignore it.
* register is suitable for frequently accessed variables when quick access is critical.

Example:

#include <stdio.h>

int main() {

register int x = 42; // 'x' is a register variable (compiler may or may not honor it)

printf("x = %d\n", x);

return 0;

}

**Extern Storage Class:**

* Variables declared as extern are used to indicate that a variable is defined in another source file.
* They are typically used for global variables shared between multiple source files.
* The actual variable is defined in another file, and the extern declaration provides access to it.

**Example (Declaration in one file):**

// File1.c

int globalVariable; // Declaration of an extern variable

**Example (Definition in another file):**

// File2.c

int globalVariable = 42; // Definition of the extern variable

In this example, globalVariable is declared as extern in File1.c, and its actual definition is in File2.c. Both source files can access and modify globalVariable

**2, Explain formatted input and output statement with examples**

Formatted input and output in C refers to the use of format specifiers to control how data is read from or written to the standard input and output streams (**stdin** and **stdout**) or other files. The **printf** function is used for formatted output, while the **scanf** function is used for formatted input. Format specifiers in both functions are preceded by the % character

**Formatted Output (printf):**

The **printf** function is used to format and display output to the console or a file.

Example 1: Displaying integers and strings

#include <stdio.h>

int main() {

int age = 25;

char name[] = "Alice";

printf("Name: %s, Age: %d\n", name, age);

return 0;

}

In this example:

* **%s** is a format specifier for strings, and it is replaced by the value of the **name** variable.
* **%d** is a format specifier for integers, and it is replaced by the value of the **age** variable.

Example 2: Formatting floating-point numbers

#include <stdio.h>

int main() {

double pi = 3.14159265359;

printf("The value of pi is approximately %.2f\n", pi);

return 0;

}

In this example:

* **%.2f** is a format specifier for floating-point numbers with two decimal places.

**Formatted Input (scanf):**

The **scanf** function is used to read and format input from the user.

Example 1: Reading integers and strings

#include <stdio.h>

int main() {

int age;

char name[50];

printf("Enter your age: ");

scanf("%d", &age); // %d is used to read an integer

printf("Enter your name: ");

scanf("%s", name); // %s is used to read a string

printf("Name: %s, Age: %d\n", name, age);

return 0;

}

In this example:

* **%d** is used in the **scanf** function to read an integer value.
* **%s** is used to read a string, but note that it does not work well for reading strings with spaces.

Example 2: Reading floating-point numbers

#include <stdio.h>

int main() {

double price;

printf("Enter the price of an item: ");

scanf("%lf", &price); // %lf is used to read a double

printf("The price you entered is: %.2lf\n", price);

return 0;

}

In this example:

* **%lf** is used in the **scanf** function to read a double-precision floating-point number.

Formatted input and output statements in C provide a way to control how data is displayed and read, ensuring proper formatting and interpretation of data. The format specifiers in these functions allow you to specify the type and format of the data being processed