Vidyavardhini College of Engineering & Technology, Mumbai, India

MCA (SEM – I)

C Programming

Assignment No.1

Q. 1 Write short notes on:

I. Keywords and Identifiers

Keywrords: Keywords are preserved words that have special meaning in C language. The meaning of C language keywords has already been described to the C compiler. These meaning cannot be changed. Thus, keywords cannot be used as variable names because that would try to change the existing meaning of the keyword, which is not allowed.(Don't worry if you do not know what variables are, you will soon understand.) There are total 32 keywords in C language.

Auto	double	int	struct
Break	Else	long	switch
Case	enum	register	typedef
Const	extern	return	union
Char	float	short	unsigned
continue	For	signed	volatile
default	goto	sizeof	void
Do	If	static	while

Identifiers: In C language identifiers are the names given to variables, constants, functions and user-define data. These identifier are defined against a set of rules.
Pre-processor commands

Rules for an Identifier:

1. An Identifier can only have alphanumeric characters (a-z , A-Z , 0-9) and underscore(_).
2. The first character of an identifier can only contain alphabet (a-z , A-Z) or underscore (_).
3. Identifiers are also case sensitive in C. For example **name** and **Name** are two different identifiers in C.
4. Keywords are not allowed to be used as Identifiers.
5. No special characters, such as semicolon, period, whitespaces, slash or comma are permitted to be used in or as Identifier.

When we declare a variable or any function in C language program, to use it we must provide a name to it, which identified it throughout the program, for example:

```
int myvariable = "Studytonight";
```

Here myvariable is the name or identifier for the variable which stores the value "Studytonight" in it.

II. getchar() and gets()

getchar(): The C library function **int getchar(void)** gets a character (an unsigned char) from stdin. This is equivalent to **getc** with stdin as its argument.

Following is the declaration for getchar() function.

- int getchar(void)

This function returns the character read as an unsigned char cast to an int or EOF on end of file or error.

The following example shows the usage of getchar() function.

```
#include <stdio.h>

int main () {
   char c;

   printf("Enter character: ");
   c = getchar();

   printf("Character entered: ");
   putchar(c);

   return(0);
}
```

gets(): The C library function **char *gets(char *str)** reads a line from stdin and stores it into the string pointed to by str. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

Following is the declaration for gets() function.

- char *gets(char *str)

Parameters:

str − This is the pointer to an array of chars where the C string is stored.

This function returns **str** on success, and **NULL** on error or when end of file occurs, while no characters have been read.

Eg.,

The following example shows the usage of gets() function.

```
#include <stdio.h>

int main () {
  char str[50];

  printf("Enter a string : ");
  gets(str);

  printf("You entered: %s", str);

  return(0);
}
```

III. putchar() and puts

putchar(): The C library function **int putchar(int char)** writes a character (an unsigned char) specified by the argument char to stdout.

Following is the declaration for putchar() function.

- int putchar(int char)

Parameters

char − This is the character to be written. This is passed as its int promotion.

This function returns the character written as an unsigned char cast to an int or EOF on error.

The following example shows the usage of putchar() function.

```
#include <stdio.h>

int main () {
  char ch;
```

```
  for(ch = 'A' ; ch <= 'Z' ; ch++) {
    putchar(ch);
  }

  return(0);
}
```

puts(): The C library function **int puts(const char *str)** writes a string to stdout up to but not including the null character. A newline character is appended to the output.

Following is the declaration for puts() function.

- int puts(const char *str)

Parameters

- **str** − This is the C string to be written.

If successful, non-negative value is returned. On error, the function returns EOF.

The following example shows the usage of puts() function.

```
#include <stdio.h>
#include <string.h>

int main () {
  char str1[15];
  char str2[15];

  strcpy(str1, "tutorialspoint");
  strcpy(str2, "compileonline");
```

```
  puts(str1);
  puts(str2);

  return(0);
}
```

IV. printf() and scanf()

printf() and scanf() functions are inbuilt library functions in C programming language which are available in C library by default. These functions are declared and related macros are defined in "stdio.h" which is a header file in C language.

We have to include "stdio.h" file as shown in below C program to make use of these printf() and scanf() library functions in C language.

printf() function in C language

- In C programming language, printf() function is used to print the "character, string, float, integer, octal and hexadecimal values" onto the output screen.
- We use printf() function with %d format specifier to display the value of an integer variable.
- Similarly %c is used to display character, %f for float variable, %s for string variable, %lf for double and %x for hexadecimal variable.
- To generate a newline,we use "\n" in C printf() statement.

Note:

C language is case sensitive. For example, printf() and scanf() are different from Printf() and Scanf(). All characters in printf() and scanf() functions must be in lower case.

EXAMPLE PROGRAM FOR C PRINTF() FUNCTION:

```
#include <stdio.h>

int main()

{

  char ch = 'A';
```

```
  char str[20] = "fresh2refresh.com";
  float flt = 10.234;
  int no = 150;
  double dbl = 20.123456;

  printf("Character is %c \n", ch);

  printf("String is %s \n" , str);

  printf("Float value is %f \n", flt);

  printf("Integer value is %d\n" , no);

  printf("Double value is %lf \n", dbl);

  printf("Octal value is %o \n", no);

  printf("Hexadecimal value is %x \n", no);

  return 0;
}
```

SCANF() FUNCTION IN C LANGUAGE:

- In C programming language, scanf() function is used to read character, string, numeric data from keyboard
- Consider below example program where user enters a character. This value is assigned to the variable “ch” and then displayed.
- Then, user enters a string and this value is assigned to the variable “str” and then displayed.

EXAMPLE PROGRAM FOR PRINTF() AND SCANF() FUNCTIONS IN C PROGRAMMING LANGUAGE:

```
#include <stdio.h>
int main()
{
  char ch;
  char str[100];

  printf("Enter any character \n");
  scanf("%c", &ch);

  printf("Entered character is %c \n", ch);
  printf("Enter any string ( upto 100 character ) \n");
  scanf("%s", &str);

  printf("Entered string is %s \n", str);
}
```

Q.2 What are data types in C? Explain in detail.

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:

Following are the examples of some very common data types used in C:

char: The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.
int: As the name suggests, an int variable is used to store an integer.
float: It is used to store decimal numbers (numbers with floating point value) with

single precision.
double: It is used to store decimal numbers (numbers with floating point value) with double precision.
Different data types also have different ranges upto which they can store numbers. These ranges may vary from compiler to compiler. Below is list of ranges along with the memory requirement and format specifiers on 32 bit gcc compiler.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	12		%Lf

We can use the sizeof() operator to check the size of a variable. See the following C program for the usage of the various data types:

```
#include <stdio.h>

int main()

{

   int a = 1;

   char b ='G';
```

```
    double c = 3.14;
    printf("Hello World!\n");

    //printing the variables defined above along with their sizes
    printf("Hello! I am a character. My value is %c and "
          "my size is %lu byte.\n", b,sizeof(char));
    //can use sizeof(b) above as well

    printf("Hello! I am an integer. My value is %d and "
          "my size is %lu  bytes.\n", a,sizeof(int));
    //can use sizeof(a) above as well

    printf("Hello! I am a double floating point variable."
          " My value is %lf and my size is %lu bytes.\n",c,sizeof(double));
    //can use sizeof(c) above as well

    printf("Bye! See you soon. :)\n");

    return 0;
}
```

Q.3 What are storage types in C programming?

Storage Classes are used to describe about the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes, namely:

auto: This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variables resides. They are assigned a garbage value by default whenever they are declared.

extern: Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well. So an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block. This basically signifies that we are not initializing a new variable but instead we are using/accessing the global variable only. The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program. For more information on how extern variables work, have a look at this link.

static: This storage class is used to declare static variables which are popularly used while writing programs in C language. Static variables have a property of preserving their value even after they are out of their scope! Hence, static variables preserve the value of their last use in their scope. So we can say that they are

initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared. Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

register: This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only. Usually few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program. An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.
To specify the storage class for a variable, the following syntax is to be followed:

Syntax:

- storage_class var_data_type var_name;

Functions follow the same syntax as given above for variables. Have a look at the following C example for further clarification:

```
// A C program to demonstrate different storage
// classes
#include <stdio.h>

// declaring and initializing an extern variable
extern int x = 9;
```

```
// declaring and initialing a global variable z
// simply int z; would have initialized z with
// the default value of a global variable which is 0
int z = 10;

int main()
{
    // declaring an auto variable (simply
    // writing "int a=32;" works as well)
    auto int a = 32;

    // declaring a register variable
    register char b = 'G';

    // telling the compiler that the variable
    // z is an extern variable and has been
    // defined elsewhere (above the main
    // function)
    extern int z;

    printf("Hello World!\n");

    // printing the auto variable 'a'
    printf("\nThis is the value of the auto "
           " integer 'a': %d\n",a);
```

```
// printing the extern variables 'x'
// and 'z'
printf("\nThese are the values of the"
      " extern integers 'x' and 'z'"
      " respectively: %d and %d\n", x, z);

// printing the register variable 'b'
printf("\nThis is the value of the "
      "register character 'b': %c\n",b);

// value of extern variable x modified
x = 2;

// value of extern variable z modified
z = 5;

// printing the modified values of
// extern variables 'x' and 'z'
printf("\nThese are the modified values "
      "of the extern integers 'x' and "
      "'z' respectively: %d and %d\n",x,z);

// using a static variable 'y'
printf("\n'y' is a static variable and its "
      "value is NOT initialized to 5 after"
      " the first iteration! See for"
```

```
        " yourself :)\n");

    while (x > 0)
    {
        static int y = 5;
        y++;

        // printing value of y at each iteration
        printf("The value of y is %d\n",y);
        x--;
    }

    // exiting
    printf("\nBye! See you soon. :)\n");

    return 0;
}
```

Q.4 What are escape characters?

In C programming language, there are 256 numbers of characters in character set. The entire character set is divided into 2 parts i.e. the ASCII characters set and the extended ASCII characters set. But apart from that, some other characters are also there which are not the part of any characters set, known as ESCAPE characters.

List of Escape Sequences

\a Alarm or Beep

\b Backspace

\f Form Feed

\n New Line

\r Carriage Return

\t Tab (Horizontal)

\v Vertical Tab

\\ Backslash

\' Single Quote

\" Double Quote

\? Question Mark

\ooo octal number

\xhh hexadecimal number

\0 Null

Example:

```
#include <stdio.h>
int main(void)
{
   printf("My mobile number "
        "is 7\a8\a7\a3\a9\a2\a3\a4\a0\a8\a");

        printf("Hello Geeks\b\b\b\bF");

        printf("Hello\n");
   printf("GeeksforGeeks");
```

```
        printf("Hello \t GFG");

        printf("Hello friends");
    printf("\v Welcome to GFG");

        printf("Hello fri \r ends");
        printf("Hello\\GFG");

        printf("\' Hello Geeks\n");

        printf("\" Hello Geeks");
        printf("\?\?!\n");

        char* s = "A\0725";
    printf("%s", s);

        char* s = "B\x4a";
    printf("%s", s);


    return (0);
}
```

Q.5 Explain prefix and postfix operator with examples (i.e., ++a & a++)

In C, precedence of Prefix ++ (or Prefix –) and dereference (*) operators is same, and precedence of Postfix ++ (or Postfix –) is higher than both Prefix ++ and *.

If p is a pointer then *p++ is equivalent to *(p++) and ++*p is equivalent to ++(*p) (both Prefix ++ and * are right associative).

For example, program 1 prints *'h'* and program 2 prints *'e'*.

```
// Program 1
#include<stdio.h>
int main()
{
  char arr[] = "geeksforgeeks";
  char *p = arr;
  ++*p;
  printf(" %c", *p);
  getchar();
  return 0;
}

// Program 2
#include<stdio.h>
int main()
{
  char arr[] = "geeksforgeeks";
  char *p = arr;
  *p++;
  printf(" %c", *p);
  getchar();
```

```
  return 0;
}
```