**Model Answer for M. C. A. First Semester,**
**Subject: Advanced Programming in C**
**SECTION –A**

**Q. No. 1.** Attempt the following (Consider header file included):

**(I) What are various rules for identifiers?**

**Ans : Rules for Identifier:**

- Any combination of alphabets, digits or underscores. But first character must be alphabets or underscore.
- No commas or blanks (White space) are allowed.
- Keywords are not allowed to use as an identifier name.
- C is case sensitive i.e. UPPER and lower case are significant.
- No Special symbol other than an underscore can be used.
- The length of identifier may be up to 31 characters but only the first 8 characters are significant by compiler.
- (Note: Some compilers allow identifier names whose length may be up to 247 characters. But, it is recommended to use maximum 31 characters in variable name. Large variable name leads to occur errors.)

**(II) What do you mean by function prototype?**

**Ans:** A function prototype or function interface in C, is a declaration of a function that omits the function body but does specify the function's return type, name, arity and argument types. In a prototype, argument names are optional (and have function prototype scope, meaning they go out of scope at the end of the prototype), however, the type is necessary along with all modifiers (e.g. if it is a pointer or a const argument).

**(III) Difference between getch() and getche():**

**Ans:** Getch() - Reads a character directly from the console without buffer, and without echo.
Getche() - Reads a character directly from the console without buffer, but with echo.

**(IV) What is the use of type def statement?**

**Ans:** The typedef keyword allows the programmer to create new names for types such as int. it literally stands for "type definition".Typedefs can be used both to provide more clarity to your code and to make it easier to make changes to the underlying data types that you use.
Ex.

```
#include<stdio.h>
#typedef int Integer
void main()
```

```
{
int a=10;
Integer b=10;
printf("a=%d, b=%d",a,b);
}
```

**(V) What is the syntax of fread() and fwrite() function?**

**Syntax of fread():** size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
**Syntax of fwrite():** size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)

**(VI). Point out the errors, if any, in the following programs:**

```
main( )
{
        int temp ;
        scanf ( "%d", &temp ) ;
        switch ( temp )
        {
           case ( temp <= 20 ) :
           printf ( "\nOoooooohhhh! Damn cool!" ) ;
           case ( temp > 20 && temp <= 30 ) :
           printf ( "\nRain rain here again!" ) ;
           case ( temp > 30 && temp <= 40 ) :
           printf ( "\nWish I am on Everest" ) ;
           default :
           printf ( "\nGood old nagpur weather" ) ;
        }
}
```

**Output:** Compile Time Error in case statement

**(VII). What will be the output of the following program?**

```
void main()
{
        int x=5,y=0,z=10;
        if(++x>5 || y++<z)
        printf("TRUE");
        printf(" x=%d, y=%d, z=%d",x,y,z);
}
```

**Output:** TRUEx=6, y=0, z=10

**(VIII). What will be the output of the following program?**

```
void main( )
{
   int i, a[ ] = { 2, 4, 6, 8, 10 } ;
   for ( i = 0 ; i <= 4 ; i++ )
   {
        i[a]=a[i]++ +i;
        printf( "%d, %d\n", a[i],*(a+i) ) ;
    }
   }
```

**Output:**

```
3,3
6, 6

9, 9

12, 12

15, 15
```

**(IX). What would be the output of the following programs:**

```
main( )
{
   int i = 5, j = 2 ;
   junk ( &i, &j ) ;
   printf ( "\n%d %d", i, j ) ;
 }
 junk ( int *i, int *j )
 {
   *i = *i * *i ;
   *j = *j * *j ;
 }
```

**Output:** 25, 4

**(X). Which of these are reasons for using pointers?**

1. To manipulate parts of an array
2. To refer to keywords such as **for** and **if**
3. To return more than one value from a function
4. To refer to particular programs more conveniently

**Ans:** **1.** To manipulate parts of an array **or 3.** To return more than one value from a function

**Q. No. 2. What is operator in C Language? Describe different types of operators with suitable example.**

**Ans:**

An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
Used in programs to manipulate data and variables
The data items that operators act upon are called operands.

**Binary Operators:** Operator that require two operands to perform mathematical operations.
**Unary Operators:** Operator that require one operands to perform mathematical operations.

C operator can be classified into a number of categories. These are:

**1. Arithmetic Operators**
**2. Relational Operators**
**3. Logical Operators**
**4. Assignment Operators**
**5. Compound Assignment Operators**
**6. Increment and Decrement Operators**
**7. Conditional Operator**
**8. Bitwise Operators**
**9. Special Operators**

**Arithmetic Operators:** C provides all the basic arithmetic operators and operates on any built-in data type allowed in C.

| **Operator** | **Meaning** | **When a=5, b=7** |
|---|---|---|
| + | Addition or unary plus | a+b=12 |
| - | Subtraction or unary minus | a-b=-2 |
| * | Multiplication | a*b=35 |
| / | Division | a/b=0, -a/-b=0, -a/b=either 0 or -1 (Machine dependent) |
| % | Modulo division | a%b=5, -a%b=-5,-a%-b=-5and a%-b=5 |

**Important Points:**

- Integer division truncates any fractional part.
- Modulo division operation produces the remainder of an integer arithmetic and cannot be used on floating point data and the sign of the result is always the sign of the first operand (dividend)
- C does not have an operator for exponentiation.

**Integer Arithmetic:** An arithmetic operation involving only real operands.

**Important Points:**

- Always yields an integer value.
- Largest integer value depend on the machine
- If both operands are of the same sign, result is truncated towards zero. if one of them is negative , the direction of truncation is implementation dependents
- 6/7=0,-6/-7=0 but -6/7 may be zero or -1 (Machine dependent)

**Real Arithmetic:** An arithmetic operation involving only real operands

**Important Points:**

- Real operand may assume values either in decimal or exponential.
- Floating point values are rounded to the number of significant digits permissible
- Final value is the approximation of the correct result.
- Operator(%) cannot be used with real operands

**Mixed-mode Arithmetic:** An arithmetic operation involving one real operand and other integer operands. The result is always real number.

**Relational Operators:** to compare two quantities and take certain decision depending on their relation. An expression containing a relational operator is termed as a relational expression.

| **Operator** | **Meaning** | **Complement** | **When a=5, b=7and c=5 (True-1, False=0)** |
|---|---|---|---|
| > | is greater than | <= | a>b=0, b>a=1 |
| >= | is greater than or equal to | < | a>=b=0,a>=c=1 c<=a=1 |
| < | is less than | >= | a<b=1, b<a=0 |
| <= | is less than or equal to | > | a<=c=1, a<=b=0 |
| == | is equal to | != | a==c=1, a==c=0 |
| != | is not equal to | == | a!=b=1, a!=c=0 |

A simple relational expression contains only one relational operator and take the following form:

arithmetic expression-1 relational operator arithmetic expression-2

**Important Points:**

- value of a relational expression is either one or zero.
- Used in decision making.

**Logical Operators:** In addition to the relational operators, C has the following three logical operators.

| **Operator** | **Meaning** |
|---|---|
| && | Logical AND |
| \|\| | Logical OR |
| ! | Logical NOT |

| **Op1** | **Op2** | **Value of Expression** | | | **When a=5, b=7and c=5** | **Result** |
|---|---|---|---|---|---|---|
| | | **op-1&&op-2** | **op-1\|\|op-2** | **!op1** | | |
| 0 | 0 | 0 | 0 | 1 | a>b&&a>c, a>b\|\|a>c | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | Non-Zero | 0 | 1 | 1 | a>=b&&a>=c,  a>b‖0 | 0 |
| Non-Zero | 0 | 0 | 1 | 0 | 5&&a>b, b<c ‖0 | 0 |
| Non-Zero | Non-Zero | 1 | 1 | 0 | b>a&&a>=c, -1‖9 | 1 |

**Important Points:**

- && and ‖ are used to combine two or more relational expression (Logical expression or a compound relational expression).
- ! are used to complement the relational expression.

**Assignment Operators:** used to assign the result of an expression to a variables.

V= expression

Where, V is Variable and expression is any Arithmetic, Relational, Logical etc. expression

**Compound Assignment Operators:** In addition to assignment C support compound (shorthand) assignment operator.

V op=  expression  <=>v= v op (expression)

Where, V is Variable, op is Operator & expression is any Arithmetic, Relational, Logical etc. expression

| **Variable** | **Operator** | **Expression** | **(Variable Operator=Expression)** |
|---|---|---|---|
| x | += | y+1 | x=x+(y+1) |
| x | -= | y>1+z | x=x-(y>1+z) |
| x | *= | z++ | x=x*(z++) |
| x | /= | y/10 | x=x/(y/10) |
| x | %= | 5/2 | x=x%(5/2) |

**Advantages:**

- What appears on the left hand side need not be repeated & therefore it becomes easier to write.
- More concise and easier to read
- more efficient

**Increment/Decrement Operators:** C allows two very useful operators not generally found in other languages and both are unary operators. These are the increment and decrement operators:

++  adds 1 to the operand and  -- subtracts 1 from the operand

| **Operator** | **Notation** | **Meaning** | **When x=10** | **Result** |
|---|---|---|---|---|
| ++ | x++ | Post increment | y=x++ | y=10,x=11 |
| | ++x | Pre increment | y=++x | y=11, x=11 |

| | | | | |
|---|---|---|---|---|
| -- | x-- | Post decrement | y=x-- | y=10, x=9 |
| | --x | Pre decrement | y=--x | y=9, x=9 |

**Post Increment/Decrement:** first assigns the value to the variable on the left and then increment the operand value.

**Pre Increment/Decrement:** first adds/subtracts 1 to the operand and then the result is assigned to the variable on left.

**Conditional (or Ternary) Operators:** C allows conditional operator"?:" to construct conditional expressions of the form:

**expression1?expression2:expression3**

The operator "?:"  work as follows: expression1 is evaluate first. If it is non-zero(true), then the expression2 is evaluated and becomes the value of the expression otherwise expression3 is evaluated and its value becomes the value of the expression.

**Bitwise Operators:** to manipulate data at the bit level and used for testing the bits, or shifting them right or left. these operators may not be applied to float or double.

| **Operator** | **Meaning** | **When x=10, y=5** | **Result** |
|---|---|---|---|
| & | Bitwise AND | x&y | 0 |
| \| | Bitwise OR | xly | 15 |
| ^ | Bitwise exclusive OR | x^y | 7 |
| << | Shift left | x<<1 | 20 |
| >> | Shift right | x>>1 | 5 |
| ~ | One's Complement | ~x | -11 |

**Special Operators:** C supports some special operators of interest such as comma operator, **sizeof** operator, pointer operators(& and *) and member selection operator(. and ->).

**Comma Operators:** used to link the related expressions together. Comma related list of expressions are evaluated left to right and value of the right-most expression is the value of the combined expression.

value=(x=5,y=10,x+y);  first assign 10 to x then assigns 5 to y and finally assigns 15 to value

**sizeof Operators:** compile time operator and when used with operand return the number of bytes the operand occupies. operand may be variable, constant or a data type qualifier. It is normally

**Important Points:**

- used to determine the lengths of arrays and structures when their size are not known to the programmer.

- used to allocate memory space dynamically to variables during execution of a program.

**Q. No. 3. Write a program to read the age of 100 persons and count the number of persons in the age group 50-60. Use for and continue statements.**

**Ans:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int i, page[100],count=0;
        for(i=0;i<10;i++)
        {
            printf("\n Enter age");
            scanf("%d",&page[i]);
        }
        for(i=0;i<100;i++)
        {
            if(page[i]>=50 && page[i]<=60)
            count++;
        }
        printf("\nPerson between age group 50-60 is %d",count);
        getch();
}
```

**Q. No. 4. Write a C program to find minimum, maximum, sum and average of the given one dimensional array.**

**Ans:**

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i, parray[10], min, max, sum=0,avg;
  for(i=0;i<10;i++)
  {
     printf("\nenter number");
     scanf("%d",&parray[i]);
  }
  min=parray[0];
  max=parray[0];
  for(i=0;i<10;i++)
  {
     sum+=parray[i];
     if(min>parray[i])
```

```
        min=parray[i];
        if(max<=parray[i])
        max=parray[i];
    }
    printf("\nMinimum number is %d",min);
    printf("\nMaximum number is %d",max);
    printf("\nSum is %d",sum);
    printf("\nAverage is %d",sum/10);

getch();
}
```

**Q. No. 5.** Define a structure data type called named **date** containing three integer members **day**, **month** and **year**. Develop an interactive modular program to perform the following tasks;

To read data into structure members by a function
To validate the date entered by another function
To print the date in the format DD/MM/YYYY by a third function.

**Ans:**

```
#include<stdio.h>
#include<conio.h>
struct date
{
int days;
int month;
int year;
};
struct date readdate()
{
        struct date tmpdate;
        /*printf("\n Enter days in two digits- ");
        scanf("%d",&tmpdate.days);
        printf("\n Enter month in two digits- ");
        scanf("%d",&tmpdate.month);
        printf("\n Enter year in four digits- ");
        scanf("%d",&tmpdate.year);*/
        printf("\n Enter Date in the DD/MM/YYYY format");
        scanf("%d/%d/%d",&tmpdate.days,&tmpdate.month,&tmpdate.year);
        return tmpdate;
}
void displaydate(struct date tmpdate)
{
  printf("Date is %d/%d/%d",tmpdate.days,tmpdate.month,tmpdate.year);
}
```

```
main()
{
    struct date mydate;
    mydate=readdate();
    displaydate(mydate);
    getch();
    return 0;
}
```

**Q. No. 6. Write short notes on the following:**
**(a) Pointer arithmetic (b) Pointer to structure (c) Malloc() function (d) Calloc() function (e) Dynamic Memory Allocation**
**Ans:**

**(a) Pointer arithmetic:** the ability to modify a pointer's target address with arithmetic operations (as well as magnitude comparisons), is restricted by the language standard to remain within the bounds of a single array object (or just after it), and will otherwise invoke undefined behavior. Adding or subtracting from a pointer moves it by a multiple of the size of the ***datatype*** it points to.

**For example,** adding 1 to a pointer to 4-byte integer values will increment the pointer by 4. This has the effect of incrementing the pointer to point at the next element in a contiguous array of integers—which is often the intended result.

The following Arithmetic operation can be performed on pointers:

**Addition:**
Pointer to Integral value allowed
Pointer to Pointer not allowed
**Subtraction:**
Pointer to Integral value allowed
Pointer to Pointer is also allowed
Any other Arithmetic operation on pointers is not allowed in C programming language.

Pointer arithmetic cannot be performed on void pointers because the void type has no size, and thus the pointed address cannot be added to.

Pointer arithmetic provides the programmer with a single way of dealing with different types: adding and subtracting the number of elements required instead of the actual offset in bytes.

**(b) Pointer to structure:** In C pointers are variables that store addresses of another variable/object/function and address can be null. Each pointer has a type it points to, called a pointer to that type, same, Structure pointer is a pointer that point to a structure variable/object.

**Example**
```
#include<stdio.h>
#include<conio.h>
struct date
{
int days;
int month;
int year;
};
main()
{
    struct date *ptrmydate, mydate;
    printf("\n Enter Date in the DD/MM/YYYY format");
    scanf("%d/%d/%d",&mydate.days,&mydate.month,&mydate.year);
    printf("Date is %d/%d/%d",mydate.days,mydate.month,mydate.year);
    ptrmydate=&mydate;
    printf("Date is %d/%d/%d",ptrmydate->days,ptrmydate->month,ptrmydate->year);
    getch();
    return 0;
}
```

**(C) malloc():** allocate a block of size bytes, return a pointer to the block (NULL if unable to allocate block)

**Syntax:** void *malloc(size_t size);

**Example:**
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  char *str;
  /* Initial memory allocation */
  str = (char *) malloc(15);
  strcpy(str, "gurughasidas");
  printf("String = %s,  Address = %u\n", str, str);
  /* Reallocating memory */
  str = (char *) realloc(str, 25);
  strcat(str, "university");
  printf("String = %s,  Address = %u\n", str, str);
  free(str);
  return(0);
}
```

**(D) calloc():** allocates the requested memory and returns a pointer to it (NULL if unable to allocate block). The difference in malloc and calloc is that malloc does not set the memory to zero where as calloc sets allocated memory to zero.

**Syntax:** void *calloc(size_t num_elements, size_t element_size);

**Example:**

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int i, n;
  int *a;
  printf("Number of elements to be entered:");
  scanf("%d",&n);
  a = (int*)calloc(n, sizeof(int));
  printf("Enter %d numbers:\n",n);
  for( i=0 ; i < n ; i++ )
  {
    scanf("%d",&a[i]);
  }
  printf("The numbers entered are: ");
  for( i=0 ; i < n ; i++ ) {
    printf("%d ",a[i]);
  }
  return(0);
}
```

**(e) Dynamic Memory Allocation:** allocating and de-allocating required memory space at runtime. Dynamic memory allocation is a pretty unique feature of C programming language amongst high level languages. It enables us to create data types and structures of any size and length to suit our programs need within the program.

Function that are used to perform Dynamic Memory Allocation in C are

**1. malloc():** Allocate a block of size bytes,return a pointer to the block (NULL if unable to allocate block)

**Syntax:** void *malloc(size_t size);

**2. calloc():** allocate a block of num_elements * element_size bytes,initialize every byte to zero, return pointer to the block (NULL if unable to allocate block)

**Syntax:** void *calloc(size_t num_elements, size_t element_size);

**3. realloc():** attempts to resize the memory block pointed to by ptr that was previously allocated with a call to malloc or calloc. If ptr is NULL, realloc is identical to malloc

**Syntax:** void *realloc(void *ptr, size_t new_size);

**4. free():** de-allocates the memory previously allocated by a call to calloc, malloc, or realloc.
**Syntax:** void free(void *ptr)

**Applications**:
1. Creating Dynamic arrays 2. Creating Dynamic data structure e.g. linked lists

**Advantage:**
1. Better utilization of memory 2. No need to allocation memory in advanced

**Q. No. 7. Write a program to compare two files specified by the user, displaying a message indicating whether the files are identical or different.**

```
#include<stdio.h>
#include<conio.h>
main()
{
    FILE *fp1,*fp2;
    char ch1,ch2;
    fp1=fopen("file1.c","r");
    if(fp1==NULL)
    {
        printf("Problem in file opening");
        exit(0);
    }
    fp2=fopen("file2.c","r");
    if(fp2==NULL)
    {

         printf("Problem in file opening");
         fclose(fp1);
         exit(0);
    }
    ch1=fgetc(fp1);
    ch2=fgetc(fp2);
    while(ch1!=EOF || ch2!=EOF)
    {
          ch1=fgetc(fp1);
          ch2=fgetc(fp2);
          if(ch1!=ch2)
          break;
          printf("%c",ch1);
     }
    if(ch1!=ch2)
    {
        printf("\nBoth file are not same");
    }
    else
    {
        printf("\nBoth file are same");
    }
    fclose(fp1);
    fclose(fp2);
    getch();
}
```