Set- 2021

1. Describe about flowchart and algorithm with suitable examples. Explain the structures of a C program. [6+6]

ans. Algorithm:

An algorithm is a step-by-step procedure for solving a problem in a finite amount of time. It is a sequence of instructions that take some input, performs some operations on it, and produce an output. An algorithm must be clear, unambiguous, and well-defined. It is generally written in plain English, pseudocode, or a programming language.

Example:

// To input two numbers and display their sum.

step 1: start
step 2: Input a and b
step 3:

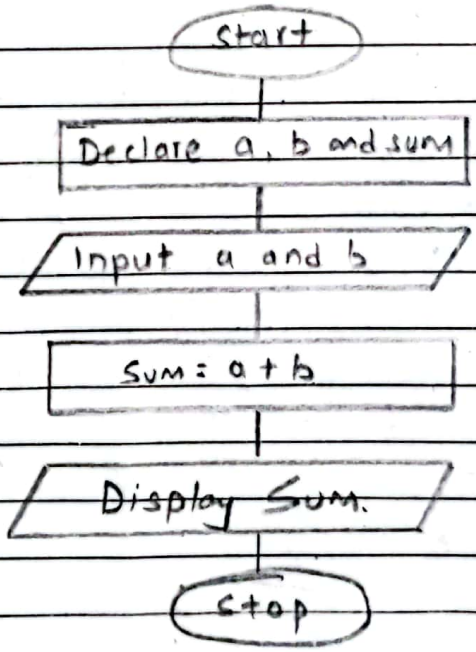
Step 1: Start
step 2: Declare a, b and sum
step 3: Input a and b
step 4: Sum = a+b
step 5: Display sum
step 6: Stop

Flowchart:

Flowchart is a graphical representation of an algorithm. It is a diagramatic representation of algorithm that shows the steps involved in solving a problem. The flowchart uses various symbols to represent the different operations performed in algorithm. The flowchart begins with a start symbols and ends with an end symbol. It also includes symbols for decision making, input/output & processing. The flowchart is used to help understand the algorithm, visualize the solution, & communicate the solution to other.

Example:

// To input two numbers and display their sum.



The advantages of using algorithm and flowchart are:

1. They help to break down complex problems into smaller, more manageable parts.
2. They provide a visual representation of the solution, making it easier to understand.
3. They help to identify errors & inefficiencies in the solution
4. They provide a common language for communicating the solution to others.

- #Structure of C programming

```
/*Documentation Section */
  Link Section
  Defination section
  Global declaration
  Main( ) function section
  {
      Declaration part
      Execution part
  }
  sub-program section
  {
      function 1
      function 2.
      function 3
  }
```

Fig: Structure of C Program

C is a group of building blocks called function which is sub-routine that may include one or more statements designed to perform at specific task.

1. Documentation Section:
- It consist of set of comment lines giving the name of program, author and other details to be used by the programmers The compiler ignores any comment so they don't add to the file size during time of execution.

2. Link Section:
- It provides instructions to compiler to link function system library.

3. Defination Section:
- It defines all symbolic constants.

4. Global Declaration section:
- It declares variables to make them global.

5. Main( ) function section:
- It is the most important section of the program & one programs contains only one main function.

- Main function start with opening brace "{" and end with closing brace "}".

- It consists declaration & execution section. Declaration

section declares all the variables used in executable part. The process to be performed is present in execution part. There must be at least one statement in executable part. Each statement ends with semi-colunn except for function defination, control statements & loops.

6. Sub-program section:

- It contains all the user defined functions that are called in main function.

4. What do you mean by dynamic memory allocation? Write a C program to determine the 2nd largest number in an array of N numbers. (Use the concept of DMA). [2+6].

ans. Dynamic Memory Allocation (DMA) in C is the process of allocating memory at runtime, i.e., when the program is executing. It is a way to allocate memory dynamically to variables when the amount of memory required is not known at compile-time. There are 4 library functions under <stdlib.h> header file. The dynamic memory allocation functions in C are: malloc( ), calloc( ), realloc( ), free( ).

i. Malloc( ):

- The malloc or 'memory allocation' method in C is used to dynamically allocate a single large block of memory with the specified size.

Syntax:

ptr = (cast type*) malloc(n * size of (data type));

calloc( ):

- used to dynamically allocate the specified number of blocks of memory of the specified type.

Syntax:

```
ptr = (cast type *) calloc (n, size of(datatype));
```

~~free( ):~~ realloc( );

- used to dynamically change the memory allocation of a previously allocated memory.

Syntax:

```
ptr = (cast type *) realloc (ptr, n * sizeof (datatype));
```

Free( );

- used to dynamically de-allocate memory.

Syntax:

```
Free(ptr);
```

// WAP to determine the 2nd largest number in an array. Using DMA.

```
#include<stdio.h>
#include<stdlib.h>
int main( )
{
    int
```

/* WAP to determine 2nd largest number in an array using DMA */

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int num=0; i=0, gt=0, temp=0, *ptr;
    printf("\nEnter how many number you want: ");
    scanf("%d", &num);
    ptr=(int *)calloc(num, sizeof(int));
    if(ptr=NULL)
    {
        printf("\n Memory cannot be allocated.");
        exit(0);
    }
    for(i=0; i<num; i++)
    {
        printf("\nEnter Numer [%d]: ", i+1);
        scanf("%d", ptr+i);
        if(*(ptr+i)>gt)
        {
            temp = gt;
            gt = *(ptr + i);
        }
        else if(*(ptr+i)>temp)
        {
            temp = *(ptr+i);
        }
```

```
        else
        {
            continue;
        }
    }
    printf("\n The second largest element is %d", temp);
    free(ptr);
    return 0;
}
```

5. Define recursion. Using recursion, write a c program to display Fibonacci series upto 1000. [2+6]

ans. Set 2022 (Q.No. 5) - About Recursion

```
#include<stdio.h>
#include<stdlib.h>
int fibo(int);
int main()
{
    int i=0, result=0;
    for(i=0; i<1000 ; i++)
    {
        result = fibo(i);
        if(result>1000)
        {
            exit(0);
        }
        printf("%d\t", result);
    }
```

```
    return 0;
}
int fibo(int n)
{
    if (n == 0)
    {
        return 0;
    }
    else if (n == 1)
    {
        return 1;
    }
    else
    {
        return fibo(n-1) + fibo(n-2);
    }
}
```

Output:

0 1 1 2 3 5 8 13 21 34 55 89 144
233 377 610 987

30. Write a program to enter name, roll-no, marks and address of 10 students and store them in structure. Display the information of students in sequence of descending order of marks. [6]

```
#include<stdio.h>
struct students
    char name[100];
    char add[100];
    int roll_no;
    float marks;
} stu[10], temp;
int main()
{
    int i=0, j=0;
    for(i=0; i<10; i++)
    {
        printf("\nEnter details of student %d:", i+1);
        printf("\n Name: ");
        fflush(stdin);
        gets(stu[i].name);
        printf("\n Address: ");
        fflush(stdin);
        gets(stu[i].add);
        printf("\n Roll No: ");
        scanf("%d", &stu[i].roll_no);
        printf("\n Marks: ");
        scanf("%f", &stu[i].marks);
    }
```

```
for(i=0; i<10; i++)
{
    for(j=i+1; j<10; j++)
    {
        if(stu[i].marks < stu[j].marks)
        {
            temp = stu[i];
            stu[i] = stu[j];
            stu[j] = temp;
        }
    }
}
printf("\n\n sequence of descending order of marks\n");
for(i=0; i<10; i++)
{
    printf("\n Rank : %d", i+1);
    printf("\n Name : %s", stu[i].name);
    printf("\n Roll No: %d", stu[i].roll_no);
    printf("\n Marks : %.1f", stu[i].marks);
    printf("\n Address : %s", stu[i].add);
}
return 0;
}
```

3.b. Write a C program to add two numeric arrays using pointers. (Elements of array must be entered by user). [6]

```
#include<stdio.h>
int main( )
{
    int a[10], b[10], i=0, *ptr, *p, sum[10], *pt;
    ptr = a;
    p = b;
    pt = sum;
    printf("\n Enter detail for 1st array: ");
    for(i=0; i<10; i++)
    {
        printf("\n Number For a[%d]: ", i);
        scanf("%d", ptr+i);
    }
    printf("\n Enter detail for 2nd array: ");
    for(i=0; i<10; i++)
    {
        printf("\n Number For b[%d]: ", i);
        scanf("%d", p+i);
    }
    printf("\n Their sum are :\n");
    for(i=0; i<10; i++)
    {
        *(pt+i) = *(ptr+i) + *(p+i);
        printf("%d\t", *(pt+i));
    }
    return 0;
}
```

6. Explain the relation between array and pointer with example. Write a C program to display factors of an input number. [3+5]

ans. Array is continuous allocation of memory or collection of data in a continuous way. A pointer is a variable which stores the memory address of another variable or array. Address of first index of array can also said to be starting address (base address) of array should assign to pointer. And, they both must have same data type. like,

```
int *ptr, num[10]
ptr = num   or   ptr = &num[0];
```

To access the value of the ~~variable~~ array that a pointer is pointing to, can use the "*" operator.
For example:

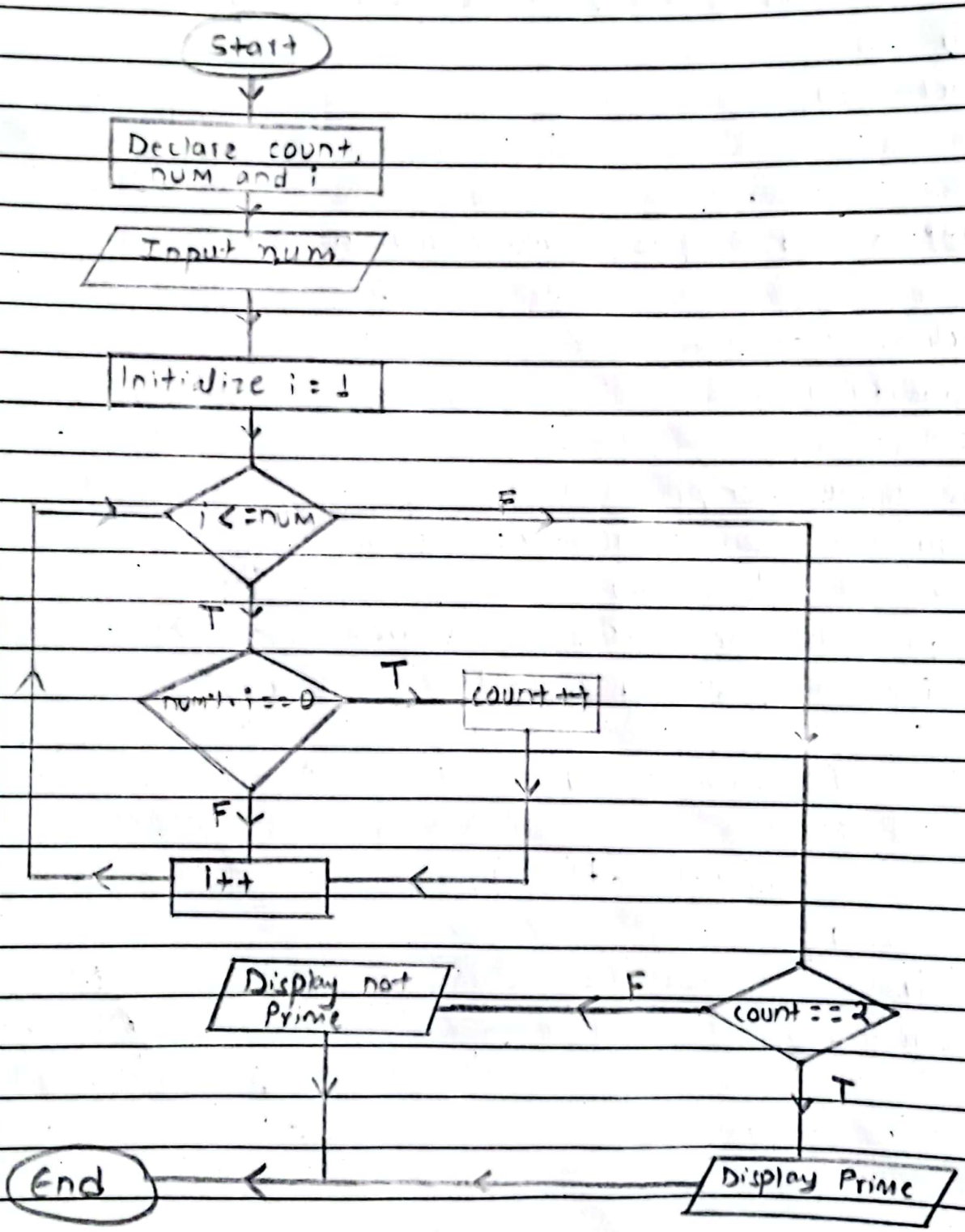
```
// To take and display elements of array using pointer.
#include<stdio.h>
int main( )
{
   int a[10], i = 0, *ptr;
   ptr = a;
   for(i = 0; i<10; i++)
   {
        printf("\n Enter a[%d] element: ", i);
        scanf("%d", &pti);
   }
```

```
for(i=0; i<10; i++)
{
    printf("\n Value of a[%d] is %d", i, *(p+i));
}
return 0;
}
```

// WAP to find factors of an input number.

```
#include<stdio.h>
int main()
{
    int num=0, i=0;
    printf("\nEnter number: ");
    scanf("%d", &num);
    printf("\nThe factors of %d are: \n", num);
    for(i=1; i<=num; i++)
    {
        if(num%i==0)
        {
            printf("%d\t", i);
        }
    }
    return 0;
}
```

7. Draw a flow chart to determine whether the entered number is prime or not. Also write a c-program for it. [4+4]



// To find a given number is prime or not.

```
#include<stdio.h>
int main()
{
    int count=0, num=0, i=0;
    printf("\n Enter your number: ");
    scanf("%d", &num);
    for(i=1; i<=num; i++)
    {
        if(num%i==0)
        {
            count++;
        }
    }
    if(count==2)
    {
        printf("\n The given number is prime.");
    }
    else
    {
        printf("\n The given number is not prime.");
    }
    return 0;
}
```

8. What are library functions? Write a c program to determine the number of vowels, consonants, digits and spaces in an ~~output~~ input string. [2+6]

ans. In C programming, a library function is a pre-written function that is included in a standard library or external library, and can be used by programmers to perform common tasks such as string manipulation, math operations, input/output operations, and more.

Library functions provide a convenient way for programmers to reuse code that has already been written and tested, which can save time and effort when developing software. They are often optimized for performance and reliability, which can result in faster and more efficient programs. Some of header file are: stdio.h, co

i. #include<stdio.h>
ii. #include<stdlib.h>
iii. #include<math.h>
iv. #include<conio.h>
v. #include<string.h>

All C standard library function are declared by using header files. We include the header files in our C program by using #include<file name.h>.

```
// To determine number of vowels, consonants, digits & space
#include<stdio.h>
int main()
{
    int space=0, vowel=0, cons=0, digit=0, i=0;
    char a[100];
    printf("\nEnter your sentences: ");
    gets(a);
    for(i=0; a[i]!='\0'; i++)
    {
        if(a[i]==32)
        {
            space++;
        }
        else if(a[i]>=65 && a[i]<=90 || a[i]>=97 && a[i]<=122)
        {
            if(a[i]=='A' || a[i]=='E' || a[i]=='I' || a[i]=='O'
            || a[i]=='U' || a[i]=='a' || a[i]=='e' || a[i]=='i'
            a[i]=='i' || a[i]=='o' || a[i]=='u')
            {
                vowel++;
            }
            else
            {
                cons++;
            }
        }
        if(a[i]>=48 && a[i]<=57){
            digit++;
        }
    }
```

```
printf("\n The total no. of space is %d", space);
printf("\n The total no. of vowels is %d", vowel);
printf("\n The total no. of consonants is %d", cons);
printf("\n The total no. of digit is %d", digit);
return 0;
}
```

10. Define variable and keyword with examples. Explain the different data types with their memory requirements and format specifiers. [2 + 6]

ans. Keywords are the reserved words in the program. That means they are defined in the C-compiler. They have special significance in c programs. They may not be used for any other purposes. Some examples of keywords are: int, float, double, struct, switch, if, return, void, for, while etc.

Variable is a container to store the data. It is a name given to the memory location where the actual data is stored. Variable names may have different data types to identify the type of value stored. For eg: int a; this means that a can store only one integer.

Where, a is a variable and int is a keyword.

- A datatype is the type of data use in the program. It is the storage representation and machine instruction to handle constant and variable. There are four kind of data types. They are:

  i. Primary Data type
  ii. User defined Datatype
  iii. Derived Datatype
  iv. Empty datatype

;. Primary data type:

- Primary datatype is such data type which doesn't use any modifiers. Primary data type can be characterized as follow:

Integers

Signed	Unsigned
int	unsigned int
short	unsigned short int
long int	unsigned long int

eg: int a=5

In signed the number can be positive or negative in result. Whereas, in unsigned the number will be positive result.

Floating point:

Float, double Float, long double

eg: Float b=5.6

character:

char

eg: char a='i', char a[ ]="Nepal"

Data type	Memory (bytes)	Range	Format specifier
short int	2	-32,768 to 32,767	%.nd
unsigned short int	2	0 to 65,535	%.nu
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 256	%c
float	4		%f
double	8		%lf
long double	16		%Lf

b. User - defined datatype:

- The user defined datatypes can be later used to declare variable. User defined data types define using type data and enum.

As an example let us consider how we can define a new type named "letter". This letter, can than be used in the same way as the predetermined datatype of c.

Syntax:

<type def><data type><new type>;

<new type> <variable>;

eg: type def int letter;

letter a;

a = 5;

Here, a new type letter is created whose data type is integer. later this letter is used as data type to declare the variable name & address.

c. Derived data type:

- Different user defined data type can be created using fundamental data types. They are array, union, structure, function are derived data types.

d. Empty data-type:

- Empty/void data-type is an empty data type that has no value and no operations. It can be used in functions and pointers.

11. Write short notes on: [4+4]

a. call by value and call by reference:

- In call by value, a copy of the actual parameter is passed to the function. This means that any changes made to the parameter inside the function do not affect the original value of the parameter. The function only operates on the copy of the parameter. It requires seperate memory location is allocated for actual and formal parameters. It is also known as passing by value.

Syntax:

```
function_name(var 1, var 2);
```

- In call by reference, the memory address of the actual parameter is passed to the function. This means that any changes made to the parameters inside the function

will affect the original value of the parameter outside the function. This done by passing a pointer to the parameter. Same memory location is allocated for actual and formal parameters. It is also known as passing by reference.

Syntax:

```
function_name(&var1, &var2);
```

Call by value and call by reference are both useful in different situations, & choosing the right approach depends on the specific requirements of the program.

b. Storage class

- In C programming, storage class specifiers are keywords that are used to define the storage allocation, scope, lifetime, and visibility of variables. The C language supports four storage classes: 'auto', 'register', 'static' and 'extern'.

i. 'auto':

The auto storage class is used by default for all local variables declared within a function. This means that the variable is allocated memory on the stack and its lifetime is limited to the scope of the function. When the function returns, the memory used by the variable is automatically deallocated.

Syntax:

auto [data_type] [variable_name];

eg: auto int number; or int number.

ii. register:

The register storage class is used to suggest to the compiler that a variable should be stored in a CPU register instead of the memory. This can make the variable access faster, but it only a suggestion and the compiler may choose to ignore it.

Syntax:

register [data_type][variable_name];

iii. static:

The static storage class is used to declare a variable that retains its value between function calls. Static variables are initialized only once and their value persists across function calls. They are allocated memory in the data segment of the program.

Syntax:

static [data_type][variable_name];

iv. extern:

The extern storage class is used to declare a variable that is defined in another file or module. This is useful when you want to share a variable across

multiple files or modules. The extern declaration tells the compiler that the variable exists somewhere else, and the linker will resolve the reference at link time.

Syntax:

```
extern [data_type] [variable_name];
```

Storage class specifiers are used to define the storage allocation, scope, lifetime, & visibility of variables in C. They help the programmer control the behaviour of variables & optimize the program's performance and memory usage.

C. Ternary Operator:

- The ternary operator is a conditional operator in C programming that allows you to write concise, one-line expressions to perform simple conditional tests. The ternary operator is also known as the conditional operator, because it allows you to test a condition and select one of two values based on the result.

Syntax:

```
condition ? value_if_true : value_if_false;
```

The condition is an expression that evaluates to a Boolean value (true or false). If the condition is true, the value_if_true expression is evaluated and returned as the result of the operator. If the condition is false, the value_if_false expression is evaluated and returned as

the result of the operator.

9: Why do we need data files? Write a C program to read the contents of a data file named "abc.doc" and copy them in uppercase to another data file named "xyz.doc". [2+6]

ans. In C programming, data files are often used to store large amounts of data that cannot be stored in memory, or data that needs to be presistent even after the program has finished executing. Some common reasons why we need data files are:

i. Storing large amounts of data:
- In some cases, the amount of data required by a program may be too large to fit in memory. Data files provide a way to store this data on disk and access it as needed.

ii. Sharing data between programs:
- Data files can be used to share data between multiple programs or processes. One program can write data to a file, and another program can read the data from same file.

iii. Persistent storage:
- Data files provides a way to store data that needs to be persistent even after the program has finished

executing. This can be useful for storing program settings, user preferences, or other types of data that need to be saved between sessions.

4. Input/Output Operations:

- Data files can be used to read input data from a file, or write output data to a file. This can be useful for tasks such as logging program output or generating reports.

Data files provide a way to store and access data that is not practical or possible to store in memory. They are an important part of many programs and can be used in a variety of ways to improve program functionality & performance.