39/45

S.No. ______ Roll No. (in Digits): 2020 [illegible]

Subject P.P.S. ______ No. of Additional Sheet(s) attached ______ Invigilator's Sign. [illegible]

Evaluation by ______ Checked By ______

Q1(a): Function: Function is defined as a set of statements which take input, do some specific computation and produce an output. Every program in C have at least one function i.e main().
syntax: return_type function_name (argument_list);

Types of function: Four types of function:
(i) Take nothing, return nothing
(2) Take something, return something
(3) Take something, return nothing.
(4) Take nothing, return something.

Function prototype: Function prototype or function declaration tells the compiler about parameter of the function, return type or data type of function and function name. Compiler check all this at the time of function defination. It is declared before main(). Prototype of function is terminated with semicolon (;)
syntax: return_type function_name (Argument list);

Page No. ......

Function defination:- Function defination contain two things i.e, function header and function ~~defination~~ body. Function header means the return type, name of function and argument that all are given at the time of declaration. While the Function body contain all the instruction that are to be performed.

syntax:- return_type function_name (Argument)

```
{
        statements 1 ;
          statements 2;
           :
      }.
```

For Ex:-

```
void  add (int ,int);    // Function prototype.
  main()
{
  add (5, 6);          // Function calling.
   getch();
  }
```

```
void add (int a, int b)      // Function defination.
{
  int z;
  z = a + b;
  printf ("%d", z)
  }
```

Program to pass an array to function & display all prime no b/w two intervals.

```
# include <stdio.h>
# include <conio.h>
void prime (int , int);
void main()
{
 int a, b;
 clrscr();
printf (" Enter any two no.:" );
 scanf ("%d %d", &a, &b);
 prime (a, b);
 getch();  }
```

~~Void prime (int x, int~~

```
# include <stdio.h>
# include <conio.h>
Void prime (int [] , int );
Void main ()
{
 int a[5] , n, i ;
  clrscr();
printf ("Enter a value:");
 scanf ("%d", & n );
printf ("Enter array elements:");
for (i=0 ; i<n; i++)
{ scanf ("%d" , & a[i]); }

 prime (a, n);
 getch();
 }
```

```
Void prime (int b[] ,int n);
{
 int i, flag = 0, m;
 m = n/2 ;
 for ( i=0; i< m; i++)
 {
  if (b[i]%i ==0)
   { flag = 1;  }.
 }.
  if (flag==0)
  { printf ("%d no is prime", b[i]);  }
   else
   { printf ("%d ~~no~~ is not prime", b[i]); }.
   }
 }.
```

8

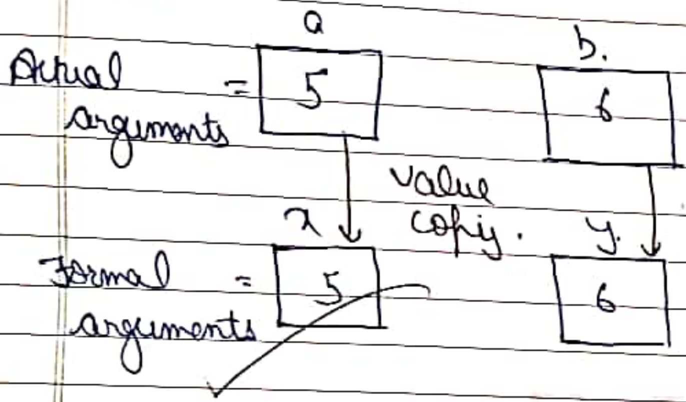
1(b) Actual argument :- Actual argument are those arguments whi
are passed to the function at the time of function
calling.
For Ex :-

```
        main( )
      {
        add (5, 6);    // actual arguments.
          getch();
          }
```

Here 5,6 are actual arguments. bcs they are passed at
the time of calling.

Formal argument :- Formal arguments are those arguments which
recieve the actual arguments at the time of function
defination. value of actual arguments are copied to
formal arguments.
For Ex :-

```
        void add (int x, int y) {    // formal arguments.
        { int z ;
            z= x+y ;
        }
```



There are two ways to pass arguments

① call by value: When the function pass only the value not any refrence. then it is called call by value. Any Change in value does not reflect to the original value.

② call by reference: when the function pass address or refrence of the variable then it is said to be a call by reference. changes in any variable reflect to the Original value.

program to swap too no :-

```
# include <stdio. h>
# include <conio. h>
void swap (int *a, int *b) ;
   main()
{
  int  a, b ;
 clrscr();
printf (" Enter two value ");
scanf ("%d %d", &a, &b);
 Swap (&a, &b);
print ("After swapping ") ;
 ~~printf ("\n %d"~~
printf (" a= %d ", a);
printf (" b= %d ", b);
    getch();
 }.

void swap (int * x, int* y)
{ ind  temp;
   temp = * x;
*x = *y ;
*y=  temp;
};
```

8

Q2 (c) Array return from function :- Array can be returned by two ways. ① Dynamically
② static.

Dynamically :-

```
int *   ara (int b[], int n).     /* int* Indicate that it will
                                     return base address */
{
 }.int  i;
   f for (i=0; i<n; i++)
   { b[i]= b[i]+5;
          }.
   return b;              // return base address of the
                                array.
   }.
```

Page No. ....

② static : when the array is declared ~~global~~ locally.

```
int *  arr()
{
static int  a[5]= {1,2,3,4,5) , i  ;    // static means that the
   for (i=0;  i<5 ; i++)                   life time of array is
   { a[i]+=5 ;                              throughout the program.
   }
                       }.
return a ;
  }.
```

Program :

```
# include <stdio.h>
# include <conio.h>
int*   even  (int [] , int n);
 void  main()
{ int  a[10] , i , *p ;
  printf (" Enter elements.");
  for  (i=0 ;i< 10 ;i++)
{
scanf ("%d", & a[i]);     8
}.

*p = even(a,10);        printf("new array");
for (i=0 ; i<10 ; i++)
{ printf ("%d", *(p+i));
  }
  getch();
  }.

int * even (~~int b[]~~ ~~[illegible]~~ int b[] , int n)
{ ~~[illegible]~~ static int c[10] , i
  for (i=0 ; i<n ; i++)
 { if (b[i] % 2 == 0)
   { c[i] = b[i] ;
        }.
   }
  return c;
  }
```

Q2(b) Dynamic memory allocation:- Dynamic memory allocation means when the memory size is ~~[illegible]~~ not known at the compile time. This memory is work on the heap. In this user can use ~~[illegible]~~ memory as per his requirements.

Four types of Dynamic memory allocation:-

1. malloc()
2. calloc()
3. realloc()
4. free()

To use all this we have to add # include <stdlib.h> header file in our program.

malloc():- Malloc() stands for memory allocation. It is a way to allocate the memory at the run time. In malloc() there is no default value, we have to initialized all the values otherwise it will take garbage value.

syntax:- ~~[illegible]~~

```
p = (cast_type*) malloc ( size of (datatype) );
```

for ex:-

```
int *p;
p = (int*) malloc (2* size of (int));
```

② calloc() : calloc() stands for contiguous allocation. If we not initialized the value then by default it will take zero. There are two arguments in calloc().
Syntax : p = (caste_type*) calloc(~~size~~ no. of elements, size of (datatype));
for ex :

```
        int *p;
    p = (int*) calloc (n*8, size of (int));
```

③ realloc() : It is used when the size to store other elements are insufficient. and to increase the size of any pre-existing data.
syntax : ~~[illegible]~~ p= (caste_type*) realloc (pointer variable name, sizeof(datatype));
for ex :

```
int *p;
    p= (int*). realloc (p, size of (int));
```

④ free() : when we initialized the variable then it is our responsibility to released it otherwise it will exist.
syntax : free (pointer variable name);

program to find largest element :

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
 Void main()
{int  *p , i , max;
clrscr();
p= (int*) calloc (10, size of (int));
 if (*p = = Null)
{ printf (" memory is full"); }
else.
{ printf (" Enter elements ");
  for (i=0 ; i<10 ; i++)
{ scanf ("%d" p+i);
   }.
}.
max = *(p+0);
```

Scanned with CamScanner

```
for (i=0; i<10; i++)
{ if (*(p+i) > max)
        max = *(p+i);

      }
printf("%d", max);
   free(p);
getch();
}
```

8