POINTERS

1. What is the output of following program?

```
# include <stdio.h>

void fun(int x)
{
x = 30;
}

int main()
{
int y = 20;
fun(y);
printf("%d", y);
return 0;
}
```

(a) 30
(b) 20
(c) Compiler Error
(d) Runtime Error

2. Output of the following program?

```
# include <stdio.h>

void fun(int *ptr)
{
  *ptr = 30;
}

int main()
{
  int y = 20;
  fun(&y);
  printf("%d", y);
return 0;
```

```
}
```

(a) 20
(b) 30
(c) Compiler Error
(d) Runtime Error

3. Output of following program?

```
#include <stdio.h>

int main()
{
   int *ptr;
   int x;
   ptr = &x;
   *ptr = 0;
   printf(" x = %d\n", x);
   printf(" *ptr = %d\n", *ptr);
   *ptr += 5;
   printf(" x  = %d\n", x);
   printf(" *ptr = %d\n", *ptr);
   (*ptr)++;
   printf(" x = %d\n", x);
   printf(" *ptr = %d\n", *ptr);
   return 0;
}
```

(a) x = 0
*ptr = 0
x = 5
*ptr = 5
x = 6
*ptr = 6

(b) x = garbage value
*ptr = 0
x = garbage value
*ptr = 5
x = garbage value
*ptr = 6

(c) x = 0
*ptr = 0
x = 5
*ptr = 5
x = garbage value
*ptr = garbage value

(d) x = 0
*ptr = 0
x = 0
*ptr = 0
x = 0
*ptr = 0

4. Consider a compiler where int takes 4 bytes, char takes 1 byte and pointer takes 4 bytes.

```
#include <stdio.h>

int main()
{
int arri[] = {1, 2 ,3};
int *ptri = arri;

char arrc[] = {1, 2 ,3};
char *ptrc = arrc;

printf("sizeof arri[] = %d ", sizeof(arri));
printf("sizeof ptri = %d ", sizeof(ptri));
printf("sizeof arrc[] = %d ", sizeof(arrc));
printf("sizeof ptrc = %d ", sizeof(ptrc));

return 0;
}
```

(a) sizeof arri[] = 3
    sizeof ptri = 4
    sizeof arrc[] = 3
    sizeof ptrc = 4

(b) sizeof arri[] = 12
    sizeof ptri = 4
    sizeof arrc[] = 3
    sizeof ptrc = 1

(c) sizeof arri[] = 3
    sizeof ptri = 4
    sizeof arrc[] = 3
    sizeof ptrc = 1

(d) sizeof arri[] = 12
    sizeof ptri = 4
    sizeof arrc[] = 3
    sizeof ptrc = 4

5. Assume that float takes 4 bytes, predict the output of following program.

```
#include <stdio.h>

int main()
{
float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
float *ptr1 = &arr[0];
float *ptr2 = ptr1 + 3;
```

```
printf("%f ", *ptr2);
printf("%d", ptr2 – ptr1);
return 0;
}
```

(a) 90.500000
3

(b) 90.500000
12

(c) 10.000000
12

(d) 0.500000
3

7.

```
#include<stdio.h>

int main( )
{
   int arr[] = {10, 20, 30, 40, 50, 60};
   int *ptr1 = arr;
   int *ptr2 = arr + 5;
   printf("Number of elements between two pointer are: %d.",
         (ptr2 - ptr1));
  printf("Number of bytes between two pointers are: %d",
         (char*)ptr2 - (char*) ptr1);
 return 0;
}
```

Assume that an int variable takes 4 bytes and a char variable takes 1 byte

(a) Number of elements between two pointers are: 5.
Number of bytes between two pointers are: 20
(b) Number of elements between two pointers are: 20.
Number of bytes between two pointers are: 20
(c) Number of elements between two pointers are: 5.
Number of bytes between two pointers are: 5
(d) Compiler Error
(e) Runtime Error

7.

```
#include<stdio.h>

int main()
{
char *x;
int a = 512;
x = (char *) &a;
x[0] = 1;
x[1] = 2;
printf("%d\n",a);
return 0;
}
```

What is the output of above program?

(a) Machine dependent
(b) 513
(c) 258
(d) Compiler Error

8.

```
#include<stdio.h>
int main()
{
  char *ptr = "ravindrababuravula";
  printf("%c\n", *&*&*ptr);
return 0;
}
```

(a) Compiler Error
(b) Garbage Value
(c) Runtime Error
(d) r

9.

```
#include<stdio.h>

void fun(int arr[])
{
```

```
int i;
int arr_size = sizeof(arr)/sizeof(arr[0]);
for (i = 0; i < arr_size; i++)
   printf("%d ", arr[i]);
}

int main()
{
  int i;
  int arr[4] = {10, 20 ,30, 40};
  fun(arr);
   }
```

(a) 10 20 30 40
(b) 10
(c) 10 20
(d) Nothing

10. The reason for using pointers in a C-program is

(a) Pointers allow different functions to share and modify their local variables.
(b) To pass large structures so that complete copy of the structure can be avoided.
(c) Pointers enable complex “linked” data structures like linked lists and binary trees.
(d) All of the above.

11.

```
#include<stdio.h>

void f(int *p, int *q)
{
  p = q;
  *p = 2;
}

int i = 0, j = 1;
int main()
{
  f(&i, &j);
  printf("%d %d \n", i, j);
```

```
  getchar();
  return 0;
}
```

(a) 2 2
(b) 2 1
(c) 0 1
(d) 0 2

12. Consider this C code to swap two integers and these five statements after it:

```
void swap(int *px, int *py)
{
   *px = *px - *py;
   *py = *px + *py;
   *px = *py - *px;
}
```

S1: will generate a compilation error
S2: may generate a segmentation fault at runtime depending on the arguments passed
S3: correctly implements the swap procedure for all input pointers referring to integers stored in memory locations accessible to the process
S4: implements the swap procedure correctly for some but not all valid input pointers
S5: may add or subtract integers and pointers.

(a) S1
(b) S2 and S3
(c) S2 and S4
(d) S2 and S5

13.

```
int f(int x, int *py, int **ppz)
{
  int y, z;
  **ppz += 1;
  z  = **ppz;
  *py += 2;
  y = *py;
  x += 3;
  return x + y + z;
```

```
}

void main()
{
  int c, *b, **a;
  c = 4;
  b = &c;
  a = &b;
  printf("%d ", f(c, b, a));
  return 0;
}
```

(a) 18
(b) 19
(c) 21
(d) 22

14. Predict the output of following program

```
#include<stdio.h>

int main()
{
    int a = 12;
    void *ptr = (int *)&a;
    printf("%d", *ptr);
    getchar();
    return 0;
}
```

(a) 12
(b) Compiler Error
(c) Run Time Error
(d) 0

15.

```
#include<stdio.h>

void swap (char *x, char *y)
{
    char *t = x;
    x = y;
```

```
    y = t;
}

int main()
{
  char *x = "ravindrababu";
  char *y = "ravula";
  char *t;
  swap(x, y);
  printf("(%s, %s)", x, y);
  t = x;
  x = y;
  y = t;
  printf("\n(%s, %s)", x, y);
  return 0;
}
```

(a) (ravindrababu,ravula)
(ravula,ravindrababu)

(b) (ravula,ravindrababu)
(ravindrababu,ravula)

(c) (ravindrababu,ravula)
(ravindrababu,ravula)

(d) (ravula,ravindrababu)
(ravula,ravindrababu)

16.

```
#include <stdio.h>

int main()
{
    int arr[] = {1, 2, 3, 4, 5};
    int *p = arr;
    ++*p;
    p += 2;
    printf("%d", *p);
    return 0;
}
```

(a) 2

(b) 3

(c) 4

(d) Compilation Error

17.

```
#include <stdio.h>

void f(char**);
int main()
{
   char *argv[] = { "ab", "cd", "ef", "gh", "ij", "kl" };
   f(argv);
   return 0;
}

void f(char **p)
{
   char *t;
   t = (p += sizeof(int))[-1];
   printf("%s\n", t);
}
```

(a) ab (b) cd
(c) ef (d) gh

18. What does the following C-statement declare?

```
int ( * f) (int * ) ;
```

(a) A function that takes an integer pointer as argument and returns an integer.
(b) A function that takes an integer as argument and returns an integer pointer.
(c) A pointer to a function that takes an integer pointer as argument and returns an integer.
(d) A function that takes an integer pointer as argument and returns a function pointer.

19.

```
#include <stdio.h>
#define print(x) printf("%d ", x)

int x;
void Q(int z)
```

```
{
   z += x;
   print(z);
  }

void P(int *y)
{
   int x = *y + 2;
   Q(x);
   *y = x - 1;
   print(x);
}

main(void)
{
   x = 5;
   P(&x);
   print(x);
}
```

The output of this program is

(a) 12 7 6
(b) 22 12 11
(c) 14 6 6
(d) 7 6 6

20.

```
#include<stdio.h>

void fun(int *p)
{
  int q = 10;
  p = &q;
}

int main()
{
  int r = 20;
  int *p = &r;
  fun(p);
```

```
  printf("%d", *p);
  return 0;
}
```

(a) 10
(b) 20
(c) Compiler error
(d) Runtime Error