Week-2 assigment

papergrid

Date: / /

(i) Predict the following output

1)
```
#include <stdio.h>
void fun (int x)
{
   x = 30;
}
int main()
{
int y = 20;
fun(y);
printf("%d", y);
return 0;
}
```

o/p = 20

2)
```
#include <stdio.h>
void fun(int *ptr)
{
*ptr = 30;
}
int main()
{
int y = 20;
fun(&y);
printf("%d", y);
return 0;
}
```

o/p :- 30

3)
```
int main()
{
  int *ptr;
  int x;
```

Date: / /

```
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;
}
```

o/p

x = 6

*ptr = 6

4)

```
#include <stdio.h>
int main()
{
char s1[7] = "1234", *p;
p = s1+2;
*p = '0';
printf("%s", s1);
}
```

o/p:- 1204

5):

```
#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;
}
```

o/p 0 2

65:-

```
#include<stdio.h>
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;
}
```

7)
```
#include <stdio.h>
int main ()
{
int arr[] = {1, 2, 3, 4, 5};
int *p = arr;
++*p;
p+ = 2;
Printf("%d", *p);
return 0;
}
```

o/p:

3

8)
```
#include <stdio.h>
int main()
{
char c[] = "GATE 2011";
char *p = c;
printf("%s", p + p[3] - p[1]);
}
```

o/p:

2011

9)
```
#include <stdio.h>
int fun (char *str1)
{
char *str2 = str1;
while(*++str1);
return (str1 - str2);
}
int main ()
{
char *str = "WookStreet";
Printf("%d", fun(str));
return 0;
}
```

o/p

10

q>

```
int main ()
{
Char arr [] = "WorkStreet";
Printf (":1.5", 9);        // 9 = .
return 0;                  printf(":1.5", &arr[4]);
}
```

o/p

Street

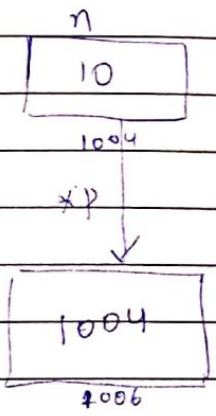
Q Explain Pointers & its operations (& and *) with proper explaination

soln:- The pointer is a variable which stores the address of the another variable.

→ The variable can be type int, char, array fun^n or any other pointer.

→ The size of the pointer depends on the architectures.

eg:-
```
int n = 10;
int *p = &n;
```



* operation :- It is used to be declare using * (asterisk symbol) It is also known as indirection pointer used to dereference a pointer.

Address of (&) operator :-

The address of operator & return the address of a variable. But we need to use %u to display the address of a variable.

eg :-
```
#include <stdio.h>
int main()
{
int number = 50;
Printf("value of the number is %d, address of number is %u", number, &number);
return 0;
}
```

o/p :-

value of the number is 50, address of the number is 1004

Q :- Explain the pointer arithemetic (++ and --)

→ We use the pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because

it is constant pointer

eg:-

```
#include <stdio.h>
const int MAX =3;

int main () {
  int var[] ={10, 100, 200};
  int i, *ptr;

ptr = var;
  for (i = 0; i< Max; i++) {
   printf("Address of var [%d] = %x\n", i, ptr);
  printf("Value of var [%d] = %d \n", i, *ptr);

   ptr ++;
  }
  return 0;
}
```

o/p:

Address of var [0] = 1004
Value of var [0] = 10
Address of var [1] = 1006
Value of var [1] = 100
Address of var [2] = 1008
Value of var [2] = 200.

Decrementing a pointer :- (--)

The same consideration apply to decrementing a pointer which decreases its value by the number of bytes of its data type

eg:
```
#include <stdio.h>
const int MAX = 3;
int main () {

  int var [] = {10, 100, 200};
  int i, *ptr;

  ptr = &var[MAX-1];

  for (i = MAX; i>0; i--) {
  Printf("Address of var [%d] = %x\n", i-1, ptr);
  Printf("Value of var [%d] = %d\n", i-1, *ptr);

   ptr--;
  }
  return 0;
}
```

o/p

```
Address of var[2] = 1006
value of var[2] = 200
Address of var[1] = 1004
value of var[1] = 100
Address of var[0] = 1002
value of var[0] = 10
```

Q Explain array name as pointer.

```
#include <stdio.h>
int main()
{
  double balance [5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
  double *p;
   int i;

P = balance;
Printf ("Array values using Pointer \n");
  for (i = 0;  i<5 ; i++)
  {
  Printf (" *(P+%d) : %f \n", i,  *(p+i));
  }
  Printf ("Array value using balance as address \n");
for (i=0 ; i<5; i++)
  {
  Printf (" *(balance"+%d) : %f \n", i, *(balance+i));
  }
   return 0;
}
```