Practice Session -II

Friday, 28 June 2024 8:26 PM

(1)

```
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));
}
```

a) 18 c) 21

b) 19 d) 24

Sol.

**ppz += 1; => c = 5

z = **ppz; => z = 5

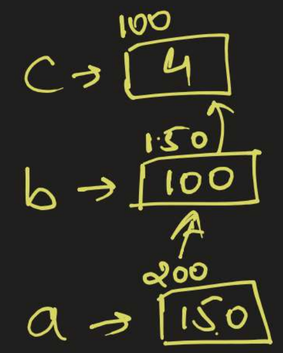
*py += 2; => c = 5 + 2 = 7

y = *py; => y = 7

x += 3; => x = 4 + 3 = 7

return x + y + z;

7 7 5



7 + 7 + 5 = 19

(2.) ISRO-16

```
void main() {
    short a = 320;
    char *ptr;
    ptr = (char *)&a;
    printf("%d", *ptr);
}
```

a.) 1 c.) 64

b.) 320 d.) error

Assumption → Running on a little-endian processor.

1234

Big-endian →

100	101	102	103
1	2	3	4

little-endian →

100	101	102	103
4	3	2	1

a = 320 →

MSB				LSB			
0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0

now a is type casted to char.
& according to little endianness.
∴ → 64

320 = most significant byte . . . least significant byte

$= 3\times10^2 + 2\times10^1 + 0\times10^0$

(3)

```
void main() {
    int num = 320, i, j = 0;

    for(i=0; i< sizeof(int)*8; i++)
    {
       if((num >> i ) & 1)
       {
          break;
       }
      j++;
    }
   printf("%d", j);
}
```

a) 4

b) ≤

c) ~~6~~

d) 7

320 → 101000000

j = ~~1~~ 6

Sol. num = $(320)_{10}$ = 0101 00000

& 1

1

j = 2

We are counting the number of zeros from LSB (Right hand side) in the binary representation of num.

④.

```
void main() {
    int n=0,a=2,b=8;
    while(n<=(a^b))
    {
        n++;
    }
    printf("%d",n);
}
```

a.) 10

~~b.) 11~~

c.) 257

d.) none

Sol.

$a = 0010$

$b = \text{XOR}, 1000$

$a \wedge b = 1010 = 10$

a	b	$a \wedge b$
0	0	0
0	1	1
1	0	1
1	1	0

bitwise operation

5.

```
struct student
{
  char name[20];
}std;

char * fun(struct student *tempStd)
{
   strcpy(tempStd->name,"Thomas");
   return tempStd->name;
}

void main() {
   strcpy(std.name,"Mike ");
   printf("%s%s",std.name,fun(&std));
}
```

a) Mike Thomas.

b) Mike Mike

c) Thomas Thomas.

d) Thomas Mike

std.name = "Mike "

std.name = "Thomas"

(6.)

```
struct employee{
   int empId;
   char *name;
   int age;
};

void main() {
   struct employee emp []={ {1,"Mike",24}, {2,"AAA",24}, {3,"BBB",25}, {4,"CCC",30} };
   printf("Id : %d, Age : %d, Name : %s", emp[2].empId, 3[emp].age, (*(emp+1)).name);
}
```

a) Id : 3, Age : 24, Name : Mike

b) Id : 3, Age : 23, Name : Mike

c) Id : 3, Age : 30, Name : AAA

d) error.

{ 3[emp] ≡ *(emp+3) ≡ emp[3] }

(7.)

```
void main() {
   union values
   {
      unsigned char a;
      unsigned char b;
      unsigned int  c;
   };

   union values val;
   val.a=1;
   val.b=2;
   val.c=300;

   printf("%d,%d,%d",val.a,val.b,val.c);
}
```

16-bit Compiler.

a) 1, 2, 300

b) 2, 2, 300

c) 44, 44, 300

d) 256, 256, 300.

union values val; → 2 Bytes.

Sol.

a = 1 → 00000000 00000001

b = 2 → 00000000 00000010

C = 300 → 00000001 00101100 → 44

1

1, 1, 300 , 44, 44, 300

Q.

```
char* fun1(void)
{
   char str[]="Hello";
   return str;
}

char* fun2(void)
{
   static char *str="Hello";
   return str;
}

void main() {
   printf("%s,%s",fun1(), fun2());
}
```

dangling pointer.

a) Hello, Hello

b) Hello, garbage

c) garbage, garbage.

d) garbage, Hello

e) error.

will be a pointer whose memory is freed.

It will be a valid pointer.

```
int * fun1(void)
{
   int x = 10;
   return &x;
}
```

dangling pointer.

(9)

```
void fun(int n)
{
  if (n > 0) {
    fun(n - 1);
    printf("%d ", n);
    fun(n - 1);
  }
}

void main() {
  fun(3);
}
```

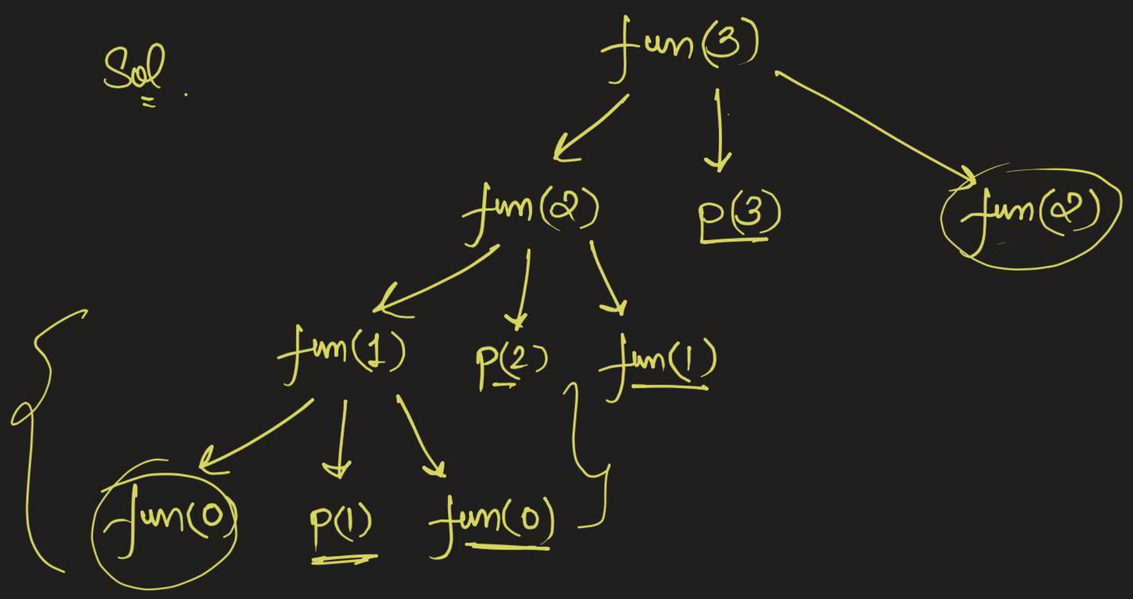
a) 1 2 3 4 3 2 1

b) 1 2 1 3 1 2 1

c) 1 2 1 4 1 2 1

d) none.

Sol.



fun(2) ← 1 2 1 3 1 2 1

10.

hw

```
int fun(int a, int b)
{
  if (b == 0)
     return 0;
  if (b % 2 == 0)
     return fun(a+a, b/2);

  return fun(a+a, b/2) + a;
}

void main() {
   printf("%d", fun(4, 3));
}
```

a) 10

b) 11

~~c) 12~~

d) 7

Function Analysis
Let's break down the function fun step by step.

fun(4, 3)

b is not 0, and b % 2 != 0 (since 3 % 2 == 1), so it goes to the return fun(a + a, b / 2) + a; branch.
Calculates fun(4 + 4, 3 / 2) + 4
Simplifies to fun(8, 1) + 4
fun(8, 1)

b is not 0, and b % 2 != 0 (since 1 % 2 == 1), so it goes to the return fun(a + a, b / 2) + a; branch.
Calculates fun(8 + 8, 1 / 2) + 8
Simplifies to fun(16, 0) + 8
fun(16, 0)

b is 0, so it returns 0.
Returning back up the recursive calls:

fun(16, 0) returns 0.
fun(8, 1) becomes 0 + 8 which is 8.
fun(4, 3) becomes 8 + 4 which is 12.
Conclusion
The final output of printf("%d", fun(4, 3)); is 12.

So, the correct answer is c) 12.

(11)

```
char str[50] = "gate 2025";
char *ptr = "hello";
printf("%d, %d", strlen(str), sizeof(str));
printf("%d, %d", strlen(ptr), sizeof(ptr));
```

a) 50, 50
b) 8, 50
c) 8, 8
d) 50, 8

a) 5, 6 c) 6, 6
b) 5, 5 d) none.

strlen → returns the no. of characters.
sizeof → returns the total occupied size.

"10"

h	e	l	l	o	\0

(7.)

```
void main()
{
    static int var;
    int j;
    for(j=0;j<=5;j+=2)
    switch(j){
      case 1:
        var++;
        break;
      case 2:
        var+=2;
      case 4:
        var%=2;
        j=-1;
        continue;
      default:
        --var;
        continue;
    }
    printf("%d",var);
}
```

a) 0

b) 1

c) 2

d) none.

Sol.

- Static variables are always initialised by 0.

var	j
0	0
-1	2
1	1
2	3
1	5
0	7

$3 <= 5$