Practice Session - I

Thursday, 27 June 2024 8:18 PM

1.

```
void main()
{
    const char var='X';
    ++var;      → compilation error.
    printf("%c",var);
}
```

a) X

b) W

c) 89

d) Y

[✓ e) none]

2.

```
void main()
{
    unsigned short var='H';  → 72
 →  var-=5;   => var = var - 5      67
    var++;  → 68
 →  printf("var : %c , %d, %d ",  var,
    var++, var);
}                 69
```

var = D

var = E

a) var : C, 67, 68

[✓ b) var : D, 68, 68]

c) var : D, 67, 67

d) var : D, 68, 69.

65	66	67	68	69	70	71	72	73
A	B	C	D	E	F	G	H	I

"%d", var

↳ 72

$x = (3 == 2) \; || \; printf("BB") ;$

BB1
AABB1

AA

$(\_\_) \; || \; (\_\_) = 1$

③

```
void main()
{
   int x;
   x= (printf("AA")|| printf("BB"));
   printf("%d",x);
   printf("\n");

   x= (printf("AA")&&printf("BB"));
   printf("%d",x);
}
```

2

1 → AA

② ②

1

a.) AABB1
AA BB1

b.) 1
1

c.) AABB1
AA1

[✓ d.)] AA1
AABB1

$|| \rightarrow \boxed{exp1} \; || \; \boxed{exp2} \; || \; exp3$

true

A or B or C

↳ ~~true~~ true if any one or more expressions are true.

atleast 1 is true.

Short circuiting or Shortcut evaluation

$\boxed{printf("AA")} = ??$

↳ ②

$\&\& \rightarrow \boxed{exp1} \; \&\& \; exp2 \; \&\& \; exp3 \ldots\ldots$

↳ false

AA1
↳ AABB1

(4.)

```
void main()
{
    int x,y;

→   x=(100,200);
→   y=100,200;

    printf("x=%d,y=%d",x,y);
}
```

a) $x = 100, \ y = 200$

b) $x = 200, \ y = 200$

[✓ c) $x = 200, \ y = 100$]

d) None.

$K = \quad x = 100, \ y = 20, \ \ldots ;$

( ) > = → operator precedence.

= > ,

(comma) "," operator is used to combine expressions & it evaluates from left to right & it always returns the rightmost expression.

$x = \boxed{(100, 200)}$

$x = (200) = 200$

$\boxed{x = 200}$

$\boxed{y = 100}, \boxed{200} \ ; \quad \Rightarrow \boxed{y = 100}$

$\boxed{x = 200, \ y = 100}$

5.

```
void main()
{
   char var=0x04;
                 ↱ bitwise OR
→  var = var | 0x04;
   printf("%d,",var);
   var |= 0x01;
   printf("%d",var);
}
```

a) 8, 9

[✓ b) 4, 5]

c) 8, 8

d) 4, 4

| →

& →

>> , << , ~

Sol.

0x04 = 0100

or 0100

0100

or 0001

0101

var = var | 0x04

var = 0x04

var = (0100) | (0001)

= 0101 = 0x05.

6

⑥

```
void main()
{
    char flag=0x0f;

    flag &= ~0x02;
    printf("%d",flag);
}
```

a) D

b) 22

[✓ c) 13]

d) 10

Sol.

$flag = 0000\,1111$

$flag = flag \;\&\; (\sim 0x02)$

$= (0000\,1111) \;\&\; (1111\,1101)$

$= 0000\,1101$

$= 0x0D = \boxed{13}$

$\sim(0000\,0010) = 1111\,1101$

$\&\; 0000\,1111$

$0000\,1101$

8.

```
void main()
{
    char cnt=0;
    for(;cnt++;printf("%d",cnt)) ;
    printf("%d",cnt);
}
```

a) Infinite loop

b) 0 1 2 127

c) 0

[✓ d) 1.]

Sol.

cnt = 0

$\boxed{\text{cnt}}$++ ⇒ false.

cnt = 0+1 = 1.

9.

a.

```
void main()
{
        int i= 5, j = 2;
        junk (i, j);
        printf ("\n%d %d", i, j);
}

junk(int i, int j) {
        i=i*i;
        j=j*j;
}
```

5, 2

25 4

a) 25 4   c) 5 4

b) 25 2   ~~d)~~ 5 2

b.

```
void main()
{
        int i= 5, j = 2 ;
        junk (&i, &j);
        printf ("\n%d %d", i, j);
}

junk(int* i, int* j) {
        i=*I * *i;
        j=*j * *j;
}
```

→ by reference.

5, 2.

$*i$

$*i = *i * *i$

$*j = *j * *j$

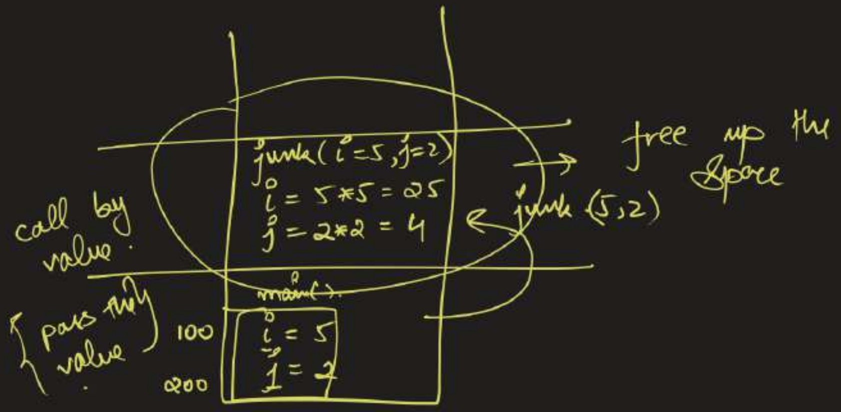
$i = 25$

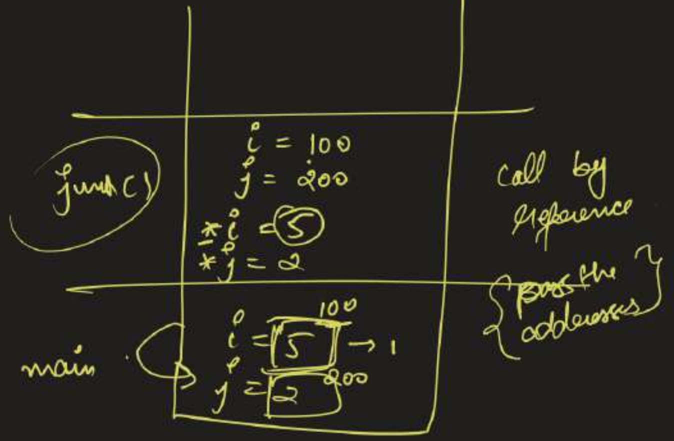
$j = 4$

$*i = 5$

$*j = 2$

$*i = 5 \times 5 = 25$





(10)

```
void main()
{
    char *str
[]={"AAAAA","BBBBB","CCCCC","DDDDD"};
    char **sptr []={str+3,str+2,str+1,str};
    char ***pp;

    pp=sptr;
    ++pp;
    printf("%s",**++pp+2);
}
```

++ > **

a) BBBB

b) CCCCC

[✓ c)] BBB

d) none

* → de-reference operator (value of operator)

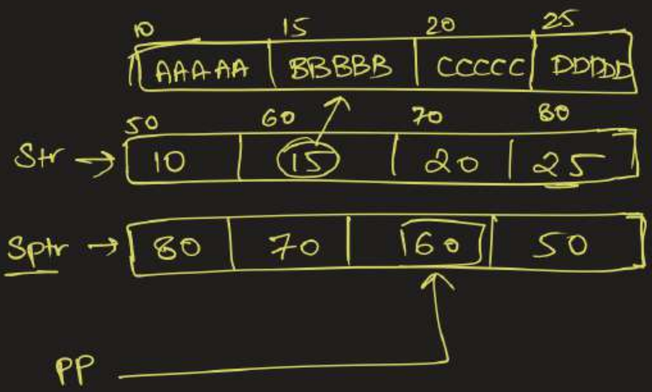
& → Reference operator (address of operator)

Sol.

*str → array pointer of strings.

** sptr → double pointer.

*** PP → Pointer which is pointing sptr's base address.



*PP → 15

**PP → BB$\boxed{\text{BBB}}$

(**PP)+2 → $\boxed{\text{BBB}}$