C Aptitude Test Question & Answers ( )

Predict the output or error(s) for the following:

```
1.          struct aaa{ struct
aaa *prev; int i;
struct aaa *next;
};

main()
{
struct aaa abc,def,ghi,jkl;
int x=100; abc.i=0;abc.prev=&jkl;
  abc.next=&def;
  def.i=1;def.prev=&abc;def.next=&ghi;
  ghi.i=2;ghi.prev=&def; ghi.next=&jkl;
  jkl.i=3;jkl.prev=&ghi;jkl.next=&abc;
x=abc.next->next->prev->next->i;
printf("%d",x);
} Answer:2
```

Explanation:

above all statements form a double circular linked list;

abc.next->next->prev->next->i

this one points to "ghi" node the value of at particular node is 2.

2. **struct point**

```
{
int x;
int y;
};
struct point origin, *pp;
main()
{
pp=&origin;
printf("origin is(%d%d)\n", (*pp).x, (*pp).y);
printf("origin is (%d%d)\n",pp->x,pp->y);
```

```
}
```

Answer:

origin is(0,0) origin

is(0,0) **Explanation:**

pp is a pointer to structure. we can access the elements of the structure either with arrow mark or with indirection operator. Note:

Since structure point is globally declared x & y are initialized as

zeroes

3. main()

{

int i=_l_abc(10);

printf(?%d\n?,?i);

}

int _l_abc(int i)

{

return(i++);

} Answer:9

Explanation:

return(i++) it will first return i and then increments. i.e. 10 will be returned.

4. main()

{

char *p; int *q;

long *r;

p=q=r=0; p++;

q++;

r++;

printf(?%p ?%p ?%p ?,p,q,r);

}

Answer: 0001 ?0002 ?0004

Explanation:

++ operator when applied to pointers increments address according to their corresponding data-types.

5. main()

```
{
char c= ? ?,x,convert(z);
getc(c);
if((c>= ?a') && (c<= ?z')) x=convert(c);
printf(?%c ?,x);
}
convert(z)
{
return z-32;
} Answer: Compiler error
```

Explanation:

declaration of convert and format of getc() are wrong.

6. main(int argc, char **argv)

```
{
printf(?enter the character ?); getchar();
sum(argv[1],argv[2]);
} sum(num1,num2) int
num1,num2;
{
return num1+num2;
}
```

Answer: Compiler error.

Explanation: *argv[1] & argv[2] are strings. They are passed to the function sum without converting it to integer values.*

7. # include *int*

```
one_d[]={1,2,3}; main()
{
int *ptr;
ptr=one_d;
ptr+=3;
printf(?%d?, *ptr);
}
```

Answer: garbage value

Explanation:

ptr pointer is pointing to out of the array range of one_d.

C Questions

Note : *All the programs are tested under Turbo C/C++ compilers. It is assumed that,*

Programs run under DOS environment,

The underlying machine is an x86 system,

Program is compiled using Turbo C/C++ compiler.

The program output may depend on the information based on this assumptions (for example sizeof(int) == 2 may be assumed).

Predict the output or error(s) for the following:

```
void main()
{
int const *p=5;
printf("%d",++(*p));
}
```

Answer: Compiler error: Cannot modify a constant value.

Explanation:

p is a pointer to a "constant integer". But we tried to change the value of the "constant integer".

```
main()
{
char s[]="man";
int i;
for(i=0;s[i];i++)
printf("\n%c%c%c%c",s[i],*(s+i),*(i+s),i[s]);
}
```

Answer:

mmmm
aaaa
nnnn

Explanation:

*s[i], *(i+s), *(s+i), i[s] are all different ways of expressing the same idea. Generally array name is the base address for that array. Here s is the base address. i is the index number/displacement from the*

base address. So, indirecting it with * is same as s[i]. i[s] may be surprising. But in the case of C it is same as s[i].

```
main()
{
float me = 1.1; double you = 1.1;
if(me==you) printf("I love U");
else
printf("I hate U");
}
```

Answer: I hate U

Explanation:

For floating point numbers (float, double, long double) the values cannot be predicted exactly. Depending on the number of bytes, the precession with of the value represented varies. Float takes 4 bytes and long double takes 10 bytes. So float stores 0.9 with less precision than long double.

Rule of Thumb:

Never compare or at-least be cautious when using floating point numbers with relational operators (== , >, <, <=, >=, !=).

```
main()
{
static int var = 5; printf("
%d ",var "); if(var)
main();
}
```

Answer: 5 4 3 2 1

Explanation:

When static storage class is given, it is initialized once. The change in the value of a static variable is retained even between the function calls. Main is also treated like any other ordinary function, which can be called recursively.

```
main()
{
int c[]={2.8,3.4,4.6.7,5};
int j, *p=c, *q=c;
```

```
for(j=0;j<5;j++) {
printf("%d ",*c);
++q;                    } for(j=0;j<5;j+
+){printf("%d ",*p);
++p;                    }
}
```

Answer : 2 2 2 2 2 2 3 4 6 5

Explanation:

Initially pointer c is assigned to both p and q. In the first loop, since only q is incremented and not c , the value 2 will be printed 5 times. In second loop p itself is incremented. So the values 2 3 4 6 5 will be printed.

```
main()
{
extern int i; i=20; printf("
%d",i);
}
```

Answer: Linker Error : Undefined symbol "_i"

Explanation:

extern storage class in the following declaration, extern int i;

specifies to the compiler that the memory for i is allocated in some other program and that address will be given to the current program at the time of linking. But linker finds that no other variable of name i is available in any other program with memory space allocated for it. Hence a linker error has occurred .

```
main()
{
int i=-1,j=-1,k=0,l=2,m;
m=i++&&j++&&k++||l++;
printf("%d %d %d %d %d",i,j,k,l,m);
}
```

Answer : 0 0 1 3 1

Explanation :

Logical operations always give a result of 1 or 0 . And also the logical AND (&&) operator has higher priority over the logical OR (||) operator. So the expression ?i++ && j++ && k++? is executed first. The result of this expression is 0 (-1 && -1 && 0 = 0). Now the expression is 0 || 2 which evaluates to 1 (because OR operator always gives 1 except for ?0 || 0? combination- for which it gives 0). So the value of m is 1. The values of other variables are also incremented by

1. main()

{

char *p;

printf(?%d%d?,sizeof(*p),sizeof(p));

}

Answer : 1 2

Explanation:

The sizeof() operator gives the number of bytes taken by its operand. P is a character pointer, which needs one byte for storing its value (a character). Hence sizeof(*p) gives a value of 1. Since it needs two bytes to store the address of the character pointer sizeof(p) gives 2. main()

{

int i=3;

switch(i)

{ default:printf(?zero?); case 1: printf(?

one?); break;

case 2:printf(?two?);

break;

case 3: printf(?three?);

break;

}

}

Answer : three

Explanation :

The default case can be placed anywhere inside the loop. It is executed only when all other cases doesn?t match.

```
main()
{
printf("%x",-1<<4);
}
```

Answer:

fff0

Explanation :

-1 is internally represented as all 1?s. When left shifted four times the least significant 4 bits are filled with 0?s.The %x format specifier specifies that the integer value be printed as a hexadecimal value.

```
main()
{
char string[]="Hello World";
display(string);
}
void display(char *string)
{
printf("%s",string);
}
```

Answer : Compiler Error : Type mismatch in redeclaration of function display

Explanation :

In third line, when the function display is encountered, the compiler doesn?t know anything about the function display. It assumes the arguments and return types to be integers, (which is the default type). When it sees the actual function display, the arguments and type contradicts with what it has assumed previously. Hence a compile time error occurs.

```
main()
{
int c=- -2;
printf("c=%d",c);
}
```

Answer : c=2;

Explanation:

Here unary minus (or negation) operator is used twice. Same maths rules applies, ie. minus *minus= plus.

Note:

However you cannot give like ?2. Because ? operator can only be applied to variables as a decrement operator (eg., i?). 2 is a constant and not a variable.

```
#define int char main()
{
int i=65;
printf(?sizeof(i)=%d?,sizeof(i));
}
```

Answer : sizeof(i)=1

Explanation:

Since the #define replaces the string int by the macro char main()

```
{
int i=10;
i=!i>14;
Printf (?i=%d?,i);
}
```

Answer : i=0

Explanation:

In the expression !i>14 , NOT (!) operator has more precedence than ? >? symbol. ! is a unary logical operator. !i (!10) is 0 (not of true is false). 0>14 is false (zero).

```
#include<stdio.h>
main()
{
char s[]={?a', ?b', ?c', ?\n?,'c?,'\0' };
char *p, *str, *str1;
p=&s[3];
str=p; str1=s;
printf(?%d?,++*p + ++*str1-32);
```

}

Answer : 77

Explanation:

p is pointing to character '\n'. str1 is pointing to character 'a' +

+*p. 'p is pointing to '\n' and that is incremented by one.' the ASCII value of '\n' is 10, which is then incremented to 11. The value of ++*p is 11. ++*str1, str1 is pointing to 'a' that is incremented by 1 and it becomes 'b'. ASCII value of 'b' is 98.

Now performing (11 + 98 '32), we get 77('M'); So we get the output 77 :: 'M'

(Ascii is 77).

#include<stdio.h>

main()

{

int a[2][2][2] = { {10,2,3,4}, {5,6,7,8} };

int *p, *q;

p=&a[2][2][2];

*q=***a;

printf("%d"-%d", *p, *q);

}

Answer : SomeGarbageValue 1

Explanation:

p=&a[2][2][2] you declare only two 2D arrays, but you are trying to access the third 2D(which you are not declared) it will print garbage values. *q=***a starting address of a is assigned integer pointer. Now q is pointing to starting address of a. If you print *q, it will print first element of 3D array.

Bitwise Operators:

Bitwise operator works on bits and perform bit by bit operation.

Assume if B = 60; and B = 13; Now in binary format they will be as

follows: A = 0011 1100

B = 0000 1101

A&B = 0000

1000

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011
Show Examples

There are following Bitwise operators supported by C language

Operator Description Example
& Binary AND Operator copies a bit to the result if it exists in both operands. (A & B)
will give 12 which is 0000 1100
| Binary OR Operator copies a bit if it exists in either operand. (A | B) will give 61 which is 0011 1101
^ Binary XOR Operator copies the bit if it is set in one operand but not both. (A ^ B) will give 49 which is 0011 0001
~ Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. (~A)
will give -60 which is 1100 0011
<< Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. A << 2 will give 240 which is 1111 0000
>> Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. A >> 2 will give 15 which is 0000 1111

Assignment Operators:

There are following assignment operators supported by C language: Show Examples

Operator Description Example
= Simple assignment operator, Assigns values from right side operands to left side operand C = A + B will assign value of A + B into C
+= Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand C += A is equivalent to C = C + A
-= Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand C -= A is equivalent to C = C - A
*= Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand C *= A is equivalent to C = C * A
/= Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand C /= A is equivalent to C = C / A
%= Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand C %= A is equivalent to C = C % A
<<= Left shift AND assignment operator C <<= 2 is same as C = C << 2
>>= Right shift AND assignment operator C >>= 2 is same as C = C >> 2
&= Bitwise AND assignment operator C &= 2 is same as C = C & 2
^= bitwise exclusive OR and assignment operator C ^= 2 is same as C = C ^ 2
|= bitwise inclusive OR and assignment operator C |= 2 is same as C = C | 2

Short Notes on L-VALUE and R-VALUE:
x = 1; takes the value on the right (e.g. 1) and puts it in the memory referenced by x. Here x and 1 are known as L-VALUES and R-VALUES respectively L-values can be on either side of the assignment operator where as R-values only appear on the right.

So x is an L-value because it can appear on the left as we've just seen, or on the right like this: y = x; However, constants like 1 are R-values because 1 could appear on the right, but 1 = x; is invalid.

Misc Operators
There are few other operators supported by C Language. Show Examples

Operator Description Example
sizeof() Returns the size of an variable. sizeof(a), where a is integer, will return 4.
& Returns the address of an variable. &a; will give actual address of the variable.
* Pointer to a variable. *a; will pointer to a variable
? : Conditional Expression If Condition is true ? Then value X : Otherwise value Y

Operators Categories:
All the operators we have discussed above can be categorized into following categories:

Postfix operators, which follow a single operand.

Unary prefix operators, which precede a single operand.

Binary operators, which take two operands and perform a variety of arithmetic and logical operations.

The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.

Assignment operators, which assign a value to a variable.

The comma operator, which guarantees left-to-right evaluation of comma-separated expressions.

Precedence of C Operators:
Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category Operator Associativity
Postfix () [] -> . ++ - - Left to right
Unary + - ! ~ ++ - - (type) * & sizeof Right to left
Multiplicative * / % Left to right
Additive + - Left to right Shift
<< >> Left to right Relational <
<= > >= Left to right Equality
== != Left to right Bitwise AND
& Left to right Bitwise XOR ^
Left to right Bitwise OR | Left
to right
Logical AND && Left to right
Logical OR || Left to right
Conditional ?: Right to left
Assignment = += -= *= /= %= >>= <<= &= ^= |= Right to left
Comma , Left to right

C - Switch Statements

1) What is the output of the following program?

```
main()
{
int I=6;
switch(I)
```

```
{ default : l+=2;
case 4:l=4;
```

```
case 5: l++;
break;
} printf("%d",l);
}
```
a)8 b)6 c)5 d)4 e)none

Answer : c) 5

2)
```
main()
{
int i=3;
switch(i)
{ default:printf("zero
"); case 1:
printf("one"); break;
case 2:printf("two");
break;
case 3: printf("three");
break;
}
}
```
Answer : three

Explanation :
The default case can be placed anywhere inside the loop. It is executed only when all other cases doesn't match.

3)
```
#include
```