

#1 Explained Report Bookmark

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

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

- **A**
Runtime Error
- **B**
Compiler Error
- **C**
5
- **D**
6

Correct Answer :B

Explanation

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"

#2Explained Report Bookmark

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

```
void main()

#include "stdio.h"

int main()

{

void *pVoid;

pVoid = (void*)0;

printf("%lu", sizeof(pVoid));

return 0;

}
```

- A

Assigning (void *)0 to pVoid isn't correct because memory hasn't been allocated. That's why no compile error but it'll result in run time error.

- B

Assigning (void *)0 to pVoid isn't correct because a hard coded value (here zero i.e. 0) can't be assigned to any pointer. That's why it'll result in compile error

- **C**
No compile issue and no run time issue. And the size of the void pointer i.e. pVoid would equal to size of int.
- **D**
sizeof() operator isn't defined for a pointer of void type.

Correct Answer :C

Explanation

(void *)0 is basically NULL pointer which is used for many purposes in C. Please note that no matter what is the type of pointer, each pointer holds some address and the size of every pointer is equal to sizeof(int). So D) isn't correct. An absolute address can be assigned to any pointer though it might result in issues at run time if the address is illegal. Since 0 is a legal address, assigning (void *)0 to pVoid is fine. So B) isn't correct. We aren't doing any illegal operation with pVoid here. So it'll not result in any compile/run time error. So A) isn't correct. For example, if we perform illegal operation over pVoid e.g. de-referencing of void pointer i.e. *pVoid, it'll result in error. The above program will compile/run without any issue. So C) is correct.

#3[Explained](#) [Report](#) [Bookmark](#)

Which of the following is not a valid variable name declaration?

- **A**
int _a3;
- **B**
int a_3;
- **C**
int 3_a;
- **D**
int _3a

Correct Answer :C

Explanation

Variable name cannot start with a digit.

Since only underscore and no other special character is allowed in a variable name

#4 [Explained](#) [Report](#) [Bookmark](#)

Variable name resolution (number of significant characters for the uniqueness of variable) depends on _____

- **A**
Compiler and linker implementations
- **B**
Assemblers and loaders implementations
- **C**
C language
- **D**
None of the mentioned

Correct Answer :A

Explanation

It depends on the standard to which compiler and linkers are adhering.

#5 [Explained](#) [Report](#) [Bookmark](#)

Which of the following is true for variable names in C?

- **A**
They can contain alphanumeric characters as well as special characters
- **B**
It is not an error to declare a variable to be one of the keywords(like goto, static)
- **C**
Variable names cannot start with a digit
- **D**
Variable can be of any length

Correct Answer :C

Explanation

According to the syntax for C variable name, it cannot start with a digit.

#6 [Explained](#) [Report](#) [Bookmark](#)

Which is a valid C expression?

- **A**
`int my_num = 100,000;`
- **B**
`int my_num = 100000;`
- **C**
`int my num = 1000;`
- **D**
`int $my_num = 10000;`

Correct Answer :B

Explanation

Space, comma and \$ cannot be used in a variable name.

#7 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{

    printf("Hello World! %d \n", x);

    return 0;

}
```

- **A**
Hello World! x;
- **B**
Hello World! followed by a junk value
- **C**
Compile time error
- **D**
Hello World!

Correct Answer :C

Explanation

It results in an error since x is used without declaring the variable x.

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{
    int y = 10000;

    int y = 34;

    printf("Hello World! %d\n", y);

    return 0;
}
```

- **A**
Compile time error
- **B**
Hello World! 34
- **C**
Hello World! 1000
- **D**
Hello World! followed by a junk value

Correct Answer :A

Explanation

Since y is already defined, redefining it results in an error.

#9 Explained Report Bookmark

Which of the following is not a valid variable name declaration?

- A
float PI = 3.14;
- B
double PI = 3.14;
- C
int PI = 3.14;
- D
#define PI 3.14

Correct Answer :D

Explanation

#define PI 3.14 is a macro preprocessor, it is a textual substitution.

#10 Explained Report Bookmark

What will happen if the following C code is executed?

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

```
int main()
```

```
{  
  
    int main = 3;  
  
    printf("%d", main);  
  
    return 0;  
  
}
```

- **A**
It will cause a compile-time error
- **B**
It will cause a run-time error
- **C**
It will run without any error and prints 3
- **D**
It will experience infinite looping

Correct Answer :C

Explanation

A c-program can have same function name and same variable name.

#11 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int main()
```

```
{  
  
    int ThisIsVariableName = 12;  
  
    int ThisIsVariablename = 14;  
  
    printf("%d", ThisIsVariablename);  
  
    return 0;  
  
}
```

- **A**
The program will print 12
- **B**
The program will print 14
- **C**
The program will have a runtime error
- **D**
The program will cause a compile-time error due to redeclaration

Correct Answer :B

Explanation

Variable names `ThisIsVariablename` and `ThisIsVariableName` are both distinct as C is case sensitive.

#12 [Explained](#) [Report](#) [Bookmark](#)

The format identifier '`%i`' is also used for _____ data type.

- A char
- B int
- C float
- D double

Correct Answer :B

Explanation

Both %d and %i can be used as a format identifier for int data type.

#13 [Explained](#) [Report](#) [Bookmark](#)

Which data type is most suitable for storing a number 65000 in a 32-bit system?

- A signed short
- B unsigned short
- C long
- D int

Correct Answer :B

Explanation

65000 comes in the range of short (16-bit) which occupies the least memory. Signed short ranges from -32768 to 32767 and hence we should use unsigned short.

#14 Explained Report Bookmark

Which of the following is a User-defined data type?

- A
typedef int Boolean;
- B
typedef enum {Mon, Tue, Wed, Thu, Fri} Workdays;
- C
struct {char name[10], int age};
- D
all of the mentioned

Correct Answer :D

Explanation

typedef and struct are used to define user-defined data types.

#15 Explained Report Bookmark

What is the size of an int data type?

- A
4 Bytes
- B
8 Bytes
- C
Depends on the system/compiler
- D
Cannot be determined

Correct Answer :C

Explanation

- Integer data type allows a variable to store numeric values.
- “int” keyword is used to refer integer data type.
- The storage size of int data type is 2 or 4 or 8 byte.
- It varies depend upon the processor in the CPU that we use. If we are using 16 bit processor, 2 byte (16 bit) of memory will be allocated for int data type.
- Likewise, 4 byte (32 bit) of memory for 32 bit processor and 8 byte (64 bit) of memory for 64 bit processor is allocated for int datatype.
- int (2 byte) can store values from -32,768 to +32,767
- int (4 byte) can store values from -2,147,483,648 to +2,147,483,647.
- If you want to use the integer value that crosses the above limit, you can go for “long int” and “long long int” for which the limits are very high.

#16 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
main()  
  
{  
  
printf("ccat");  
  
main();  
  
}
```

- **A**
Wrong statement

- **B**
It will keep on printing ccat
- **C**
It will Print ccat once
- **D**
None of the mentioned

Correct Answer :B

Explanation

In this program, the main function will call itself again and again. Therefore, it will continue to print ccat.

#17 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{
    signed char chr;

    chr = 128;

    printf("%d\n", chr);

    return 0;
}
```

- **A**
-128
- **B**
128
- **C**
Error
- **D**
depends on the compiler

Correct Answer :A

Explanation

signed char will be a negative number.

#18 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    unsigned char chr;
    chr = 128;
    printf("%d\n", chr);
    return 0;
}
```

- **A**
128
- **B**
-128
- **C**
error
- **D**
output depends on the compiler

Correct Answer :A

Explanation

unsigned char will be a positive number.

#19 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    signed char chr;

    chr = 128;

    printf("%f", chr);

    return 0;
}
```

- **A**
0.000000
- **B**
-128
- **C**
compiler error
- **D**
runtime error

Correct Answer :A

Explanation

| printf("%f", chr); %f will print the default value of double thats 0.000000

#20 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{

    float x = 'a';

    printf ("%f", x);

    return 0;

}
```

- **A**
a
- **B**
run time error
- **C**
a.0000000
- **D**
97.000000

Correct Answer :D

Explanation

Since the ASCII value of a is 97, the same is assigned to the float variable and printed.

#21 [Explained](#) [Report](#) [Bookmark](#)

Which of the data types has the size that is variable?

- **A**
int
- **B**
struct
- **C**
float
- **D**
double

Correct Answer :B

Explanation

Since the size of the structure depends on its fields, it has a variable size.

#22 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{

    enum {ORANGE = 5, MANGO, BANANA = 4, PEACH};

    printf("PEACH = %d\n", PEACH);

}
```

- A
PEACH = 3
- B
PEACH = 4
- C
PEACH = 5
- D
PEACH = 6

Correct Answer :C

Explanation

In enum, the value of constant is defined to the recent assignment from left.

#23 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

#define a 10

int main()

{

    const int a = 5;

    printf("a = %d\n", a);

}
```

- **A**
a = 5
- **B**
a = 10
- **C**
Compilation error
- **D**
Runtime error

Correct Answer :C

Explanation

The #define substitutes a with 10 without leaving any identifier, which results in Compilation error.

when evaluate it will look like const int 10 = 5; which is valid identifier hence you will get

error: expected identifier or '(' before numeric constant

#24 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int var = 010;

    printf("%d", var);
}
```

- A 2
- B 8
- C 9
- D 10

Correct Answer :B

Explanation

010 is octal representation of 8.

#25 Explained Report Bookmark

What will be the output of the following C code?

```

#include <stdio.h>

enum birds {SPARROW, PEACOCK, PARROT};

enum animals {TIGER = 8, LION, RABBIT, ZEBRA};

int main()

{

    enum birds m = TIGER;

    int k;

    k = m;

    printf("%d\n", k);

    return 0;
}

```

- **A**
0
- **B**
Compile time error
- **C**
1
- **D**
8

Correct Answer :D

Explanation

m is an integer constant, hence it is compatible.

#26 [Explained](#) [Report](#) [Bookmark](#)

Which of the following is not a C memory allocation function

- **A**
alloc()
- **B**
calloc()
- **C**
free
- **D**
malloc()

Correct Answer :A

Explanation

Dynamic Memory Allocation in C using malloc(), calloc(), free() and realloc() only

#27 [Explained](#) [Report](#) [Bookmark](#)

All memory management functions are found in

- **A**
stdlib.h
- **B**
stdio.h
- **C**
conio.h
- **D**
math.h

Correct Answer :A

Explanation

C provides some functions to achieve these tasks. There are 4 library functions provided by C defined under `<stdlib.h>` header file to facilitate dynamic memory allocation in C programming. They are:

1. `malloc()`
2. `calloc()`
3. `free()`
4. `realloc()`

#28 [Explained](#) [Report](#) [Bookmark](#)

The function that returns memory to the heap is

- A `alloc()`
- B `free()`
- C `malloc()`
- D `realloc()`

Correct Answer :B

Explanation

Free function to clear the *memory to the heap* and it will reduce the *memory access time*.

#29 [Explained](#) [Report](#) [Bookmark](#)

Among 4 header files, which should be included to use the memory allocation functions like malloc(), calloc(), realloc() and free()?

- **A**
#include<string.h>
- **B**
#include<memory.h>
- **C**
. Both b and a
- **D**
#include<stdlib.h>

Correct Answer :D

Explanation

#include <stdlib.h> is a header filer, which contains the inbuilt functions for all memory allocation functions.

#30 [Explained](#) [Report](#) [Bookmark](#)

The syntax of free() function

- **A**
void free(void* ptr)
- **B**
int free(void* ptr)
- **C**
float free(void* ptr)
- **D**
void free(ptr)

Correct Answer :A

Explanation

- free() deallocates a memory block which are all allocated previously by malloc, calloc and realloc.
- free() is mainly used when the data in the memory blocks will no longer required.

The syntax for the free function in the C Language is: void free(void *ptr);

(ptr)The pointer to the memory block to release.

Note :The free function does not return anything.

#31 [Explained](#) [Report](#) [Bookmark](#)

Which of the memory function allocates a block of memory

- A
malloc
- B
calloc
- C
release
- D
free

Correct Answer :A

Explanation

The <stdlib.h> library has functions responsible for dynamic memory management.

Function	Purpose
----------	---------

malloc

Allocates the memory of requested size and returns the pointer to the first byte of allocated space.

calloc

Allocates the space for elements of an array. Initializes the elements to zero and returns a pointer to the memory.

realloc

It is used to modify the size of previously allocated memory space.

Free

Frees or empties the previously allocated memory space.

#32 Explained Report Bookmark

Return type of a realloc() function is

- A int
- B float
- C char
- D void

Correct Answer :D

Explanation

The syntax for the realloc function in the C Language is:

```
void *realloc(void *ptr, size_t size);
```

#33 [Explained](#) [Report](#) [Bookmark](#)

Return type of a calloc() function is

- A void
- B int
- C float
- D char

Correct Answer :A

Explanation

The syntax for the calloc function in the C Language is:

```
void *calloc(size_t num_members, size_t size);
```

#34 [Explained](#) [Report](#) [Bookmark](#)

Which of the following memory management function used to release memory

- A malloc
- B calloc
- C release

- **D**
free

Correct Answer :D

Explanation

The free() function is called to release/deallocate memory. By freeing memory in your program, you make more available for use later.

#35 [Explained](#) [Report](#) [Bookmark](#)

Which of the following is not a standard file stream?

- **A**
stdin
- **B**
stderr
- **C**
stdfile
- **D**
stdout

Correct Answer :C

Explanation

Predefined Streams :

STDIN	STANDARD INPUT
--------------	---------------------------

stdout	Standard Output
stderr	Standard Error

#36 Explained Report Bookmark

The C library that contains the prototype statements for the file operations is

- **A**
file.h
- **B**
proto.h
- **C**
stdio.h
- **D**
stdlib.h

Correct Answer :C

Explanation

`void *malloc(size_t size)` : Allocates the requested memory and returns a pointer to it.

`void *calloc(size_t nitems, size_t size)`: Allocates the requested memory and returns a pointer to it.

#37 Explained Report Bookmark

In C, local variable are stored in

- A stack
- B heap
- C permanent storage
- D hard disk

Correct Answer :A

Explanation

Local variables are usually stored on the stack, and global variables in a program's "text" segment (in the case of string constants) or on the heap if they're dynamically allocated.

#38 [Explained](#) [Report](#) [Bookmark](#)

The linked list field(s) are

- A data
- B pointer
- C pointer to next node
- D data and pointer to next node

Correct Answer :D

Explanation

A linked list node contains two fields one is data and other one contains pointer to next node

#39 [Explained](#) [Report](#) [Bookmark](#)

Dynamic memory area is

- A Heap
- B stack
- C permanent storage
- D Hard disk

Correct Answer :A

Explanation

The heap is a memory used by programming languages to store global variables. By default, all global variable are stored in heap memory space. It supports Dynamic memory allocation. The heap is not managed automatically for you and is not as tightly managed by the CPU

#40 [Explained](#) [Report](#) [Bookmark](#)

The contents of the storage space allocated dynamically, can be accessed through

- A structure variables
- B pointers
- C unions

- **D**
arrays

Correct Answer :B

Explanation

In C programming memory is allocated using the function malloc() and calloc(). On Successful execution this returns the starting address of the block of dynamically allocated memory. The starting address is stored in pointers variable , hence pointers are used to access the contents of the storage space allocated dynamically.

#41 [Explained](#) [Report](#) [Bookmark](#)

In C, program instructions are stored in

- **A**
stack
- **B**
heap
- **C**
permanent storage
- **D**
Hard disk

Correct Answer :C

Explanation

A program is a sequence of instructions stored in main memory/permanent memory. When a program is run, the CPU fetches the instructions and executes or follows the instructions.

#42 Explained Report Bookmark

In C, Global variables are stored in

- **A**
Data Segment
- **B**
stack
- **C**
heap
- **D**
Hard disk

Correct Answer :A

Explanation

You got some of these right, but whoever wrote the questions tricked you on at least one question:

- global variables -----> data
- static variables -----> data
- constant data types -----> code and/or data. Consider string literals for a situation when a constant itself would be stored in the data segment, and references to it would be embedded in the code
- local variables(declared and defined in functions) -----> stack
- variables declared and defined in `main` function -----> stack
- pointers(ex: `char *arr, int *arr`) -----> - data or stack, depending on the context. C lets you declare a global or a `static` pointer, in which case the pointer itself would end up in the data segment.
- dynamically allocated space(using `malloc, calloc, realloc`) -----> heap

It is worth mentioning that "stack" is officially called "automatic storage class".

#43 [Explained](#) [Report](#) [Bookmark](#)

In C, static variables are stored in

- A permanent storage
- B heap
- C Hard disk
- D Stack

Correct Answer :A

Explanation

Global variables, `static` variables and program instructions get their memory in permanent storage area whereas local variables are stored in a memory area called Stack.

#44 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int main( void )
```

```
{
```

```

int val1 = 0x64;

int val2 = 064 + val1;

int val3 = 0x72 + 072 + 72 - val1 + val2;

printf("val2=%d val3=%d\n",val2,val3);

return 0;

}

```

- **A**
val2=152 val3=296
- **B**
val2=150 val3=298
- **C**
val2=148 val3=288
- **D**
val2=148 val3=288

Correct Answer :A

Explanation

```

#include<stdio.h>

int main( void )

{

int val1 = 0x64;

//Val1 =0x64 conversion to decimal value - 100

```

```

//064 coversion to decimal will give as value 52

int val2 = 064 + val1;

//val2 = holds a addition of 100 +52 = 152

int val3 = 0x72 + 072 + 72 - val1 + val2;

//similar 0x72 decimal conversion value is 114

// similar =0x72 decimal conversion value is 58

//so val3 = 114 +58 +72-100+152

//val3 = 244-100+152

//val3 = 144+152

//val3 = 296

printf("val2=%d val3=%d\n",val2,val3);

return 0;

}

```

#45[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include <stdio.h>

int main( void )

```

```

{
    int num1,num2,num3;

    num1 = 144;

    num2 = 156;

    num3 = printf("%10d",++num1 )+ ++num2;

    printf(" %d",num3);

    return 0;
}

```

- **A**
144 166
- **B**
145 167
- **C**
145 168
- **D**
144 167

Correct Answer :B

Explanation

num1= 144

++num1 =>145

but format specifier "%10d" will give number of characters which is 10 so

`10 + ++num2 ==> 10 + 157`

`= 167`

Output will be 145 167

#46 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )
{
    int i = 100;
    int j = 200;
    printf("%d", ++( i + j ) );
    return 0;
}
```

- **A**
300
- **B**
301
- **C**
302
- **D**
Compile Time Error

Correct Answer :D

Explanation

```
error: lvalue required as increment operand
```

In C/C++ the pre-increment (decrement) and the post-increment (decrement) operators require an L-value expression as operand. Providing an R-value or a *const* qualified variable results in compilation error.

In the above program, the expression `-i` results in R-value which is operand of pre-increment operator. The pre-increment operator requires an L-value as operand, hence the compiler throws an error.

The increment/decrement operators needs to update the operand after the sequence point, so they need an L-value. The unary operators such as `-`, `+`, won't need L-value as operand. The expression `-(++i)` is valid.

#47 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )
{
    int val1, val2, val3;
```

```

val1 = 11, (22), 33;

val2 = ((44 , 55) , 66) ;

val3 = (77) , 88 , 99 ;

printf("val1=%d val2=%d val3=%d", val1, val2, val3);

return 0;

}

```

- **A**
val1=11 val2=66 val3=77
- **B**
val1=33 val2=66 val3=99
- **C**
val1=11 val2=44 val3=77
- **D**
val1=33 val2=44 val3=99

Correct Answer :A

Explanation

If more than one operators are involved in an expression, C language has a predefined rule of priority for the operators.

In C, precedence of arithmetic operators(*, %, /, +, -) is higher than relational operators(==, !=, >, <, >=, <=) and precedence of relational operator is higher than logical operators(&&, || and !).

#48 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )
{
    printf("\n %.2f", sizeof('A' + 'a') / 8.0f);

    return (0);
}
```

- A
0.50
- B
1.00
- C
0.12
- D
0.25

Correct Answer :A

Explanation

size of(int) is 4

But `sizeof('A' + 'a')/8.0f` means explicit type casting is done

'A' and 'a' are characters so it taking ASCII value corresponding to given char

sizeof (65 + 97) / 8.0 f

Sizeof (162) / 8.0 f

Sizeof (int) / 8.0f

4 / 8.0 f

0.5

but format specifier is declared as %.2f that's

Means 2 numbers after decimal so ans is 0.50

#49 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )
{
    int num1 = 0, num2 = -1 , num3 = -2, num4 = 1, ans;

    ans = num1++ && num2++ || ++num4 && num3++;

    printf("%d %d %d %d",num1, num2, num3, num4, ans);
```

```
    return 0;  
}
```

- A
0 0 -1 2 0
- B
0 -1 -1 2 1
- C
1 0 -1 2 0
- D
1 -1 -1 2 1

Correct Answer :D

Explanation

&& is logical operator hence if first condition is false it will not check another

Hence num2++ will not get executed

so num1++ = > 1

num2++ ==> condition not executed so value remain unchanged

then ++num4 ==> incremented by 1 so became 2 and condition is satisfy so next statement is executed

so num3++ become -1

so final output will be

num1 =1

num2 = -1

num3 = -1

num4 =2

ans - 1

#50 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )
{
    int val=0;

    ++val && ++val==1 && --val;

    printf(" val=%d ", val);

    return 0;
}
```

- **A**
val=3
- **B**
val=2

- **C**
val=1
- **D**
val=0

Correct Answer :B

Explanation

```
int main( void )  
  
{  
  
    int val=0;  
  
    ++val && ++val==1 && --val;  
  
    //++val --> it will increase the value by 1 then val =1  
  
    //again ++val will increase by 1 now the value become val = 2  
  
    //so once it compare it 1 result will false so no next  
    condition will be check  
  
    printf(" val=%d ", val);  
  
    //Final output will be 2  
  
    return 0;  
  
}
```

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )

{
    int x1=1, x2=2, x3=3;

    int val=!((x1+x2)<(x3-1));

    printf(" val=%d ", !val);

    return 0;
}
```

- **A**
val=0
- **B**
val=1
- **C**
val=2
- **D**
garbage value

Correct Answer :A

Explanation

```
#include<stdio.h>

int main( void )
```

```

{

    int x1=1, x2=2, x3=3;

    int val=!((x1+x2)<(x3-1));

    // (x1+x2) ==> 3

    // x3-1==>2

    // (x1+x2)<(x3-1) ==> condition will false but we have

    // ! sign that will convert faslse to true so x3 stores 1

    printf(" val=%d ", !val);

    //While printing again we are converting true to false
    so

    // false == 0 it will print 0 as output

    return 0;

}

```

#52 Explained Report Bookmark

What will be the output of the following C code?

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

```
int main( void )
```

```

{
    int var1=10, var2=20;

    var2-= var1--;
    printf("var2=%d var1=%d", var2, var1);

    return 0;
}

```

- **A**
var2=11 var1=9
- **B**
var2=10 var1=9
- **C**
var2=11 var1=10
- **D**
var2=10 var1=10

Correct Answer :B

Explanation

```

#include<stdio.h>

int main( void )
{
    int var1=10, var2=20;

    var2-= var1--;

```

```
//var2 = 20-10 so var2 has 10 as value

//then var1-- executed and updated value as 9

printf("var2=%d var1=%d",var2, var1);

//final output will be 10 9

return 0;

}
```

#53 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )

{
    int num1 = 1, num2 = 0, num3 = 5;

    int ans1 = num1 && num2++ && num3++;

    int ans2 = --num1 || ++num2 && num3++;

    printf("num2=%d num3=%d ", num2,num3);

    printf("ans1-ans2=%d ", ans1-ans2);

    return 0;
}
```

}

- A
num2=1 num3=7 ans1-ans2= 1
- B
num2=1 num3=6 ans1-ans2=-1
- C
num2=2 num3=7 ans1-ans2= 0
- D
num2=2 num3=6 ans1-ans2=-1

Correct Answer :D

Explanation

&& has left to right associativity. Therefore num1 will be evaluated first and as it is true then next condition will be evaluated and it turns out to be false because num2 is 0. Due to post increment in num2++, num2 will become 1 after using in the expression. As num2 is 0 and condition became false then there is no need to check further and ans1 is assigned with 1.

(--num1) || (++num2 && num3++) is the correct way of understanding given expression in the problem. First of all && will be evaluated and this turns out to be true. Therefore there is no need to evaluate --num1.

#54 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )

{
```

```

int i = 0, j=0;

j = !( !( ++i && (i++ == 1)) );

printf("\n i=%d j=%d", i, j);

return 0;

}

```

- **A**
i=2 j=1
- **B**
i=1 j=1
- **C**
i=2 j=0
- **D**
i=1 j=1

Correct Answer :A

Explanation

J= !(!(++i && (i++ == 1)));

= !(!(1 && (i++ == 1)));

Here , i has post increment operator so first the condition will be checked
then i will increment by 1

so it incremented by 2 times so value of i became i =2

= !(!(1)) ; // && both operand are true

= !(0);

J = 1

So j=1,i=2

#55 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )
{
    int num1 = 0;
    float num2=100.9999f;
    double num3=200.8888;
    printf("%.6f", (float)sizeof(num1+num2+num3)/1.0f);

    return 0;
}
```

- **A**
16.000000
- **B**
12.000000
- **C**
08.000000

- D
8.000000

Correct Answer :D

Explanation

(float)Sizeof(double)/1.0f

Size of double is 8 so

(Float) 8/1.0f(Float)8

Format specifier format the value upto 6 decimal places

% .6d ==> 8.000000

Note:- by default any no. Is consider as double

#56 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )
{
    int num1, num2, num3;
    num1 = 10 / 2 + 5 * 3;
```

```

num2 = num1++ * 5;

num3 = ++num2 / 2;

printf("\n num1=%d num2=%d num3=%d", num1, num2, num3);

return 0;

}

```

- **A**
num1=21 num2=101 num3=50
- **B**
num1=21 num2=105 num3=52
- **C**
num1=20 num2=100 num3=50
- **D**
num1=20 num2=101 num3=51

Correct Answer :A

Explanation

```

#include <stdio.h>

int main( void )

{

    int num1, num2, num3;

    num1 = 10/ 2 + 5 *3;

    //both / and * has same precedence so

```

```

//10/ 2 => 5

//5 *3 => 15 then addition will happen

//5 +15 = 20

num2 = num1++ * 5;

//first num1 = 20 and multiply by 5 then afterwards its
value

//became 21 so num 2 =>100 and num1 = 21

num3 = ++num2 / 2;

// now ++num2 value is increased by 1 because it has
pre increment operator

//now num3 = 50 because num3 has data type as integer it
holds value

///without decimal part

printf("\n num1=%d num2=%d num3=%d", num1, num2, num3);

// final output will be num1=21 num2=101 num3=50

return 0;

}

```

#57[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include <stdio.h>

int main( void )
{

    int val1 = 16, val2 , val3, ans;

    val2= ~~val1;

    val3= ~val2, val2+1;

    ans= val1 + val2+ val3;

    printf("\n ans=%d", ans);

    return 0;
}

```

- **A**
ans = 15
- **B**
ans = 48
- **C**
ans = 17
- **D**
ans = 16

Correct Answer :A

Explanation

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

```

int main( void )

{

    int val1 = 16, val2 , val3, ans;

    val2= ~~val1; // val 2 => ~~16

    val3= ~val2, val2+1; // ~(16) => -17

    ans= val1 + val2+ val3;

    //now we have val1 =16 val2 = 16 and val3 =-17 then

    // 16 + 16 - 17 => 15

    printf("\n ans=%d", ans);

    return 0;

}

```

#58 Explained Report Bookmark

What will be the output of the following C code?

```

#include <stdio.h>

int main( void )

{

```

```

printf("\n ans1=%d ans2=%x ans3=%o", 0100,0100,0100);

return 0;

}

```

- **A**
ans1=64 ans2=40 ans3=100
- **B**
ans1=100 ans2=100 ans3=0100
- **C**
ans1=64 ans2=64 ans3=0100
- **D**
ans1=40 ans2=40 ans3=0100

Correct Answer :A

Explanation

octal to decimal conversion => $0100 = (0 \times 8^3) + (1 \times 8^2) + (0 \times 8^1) + (0 \times 8^0) = 64$

ans1=64

$$100_8 = 1 \cdot 8^2 + 0 \cdot 8^1 + 0 \cdot 8^0 = 64 + 0 + 0 = 64_{10}$$

Happened: **64₁₀**

Converting 64₁₀ in Hexadecimal system here so:

The whole part of a number is obtained by dividing on the basis new

64

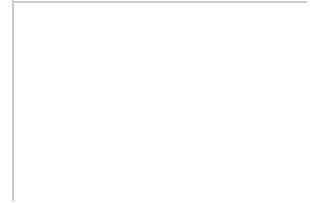
16

-64

4

0

←
direction of gaze



Happened: $64_{10} = 40_{16}$

Result of converting:

$100_8 = 40_{16}$

ans2=40

ans3 = 100 octal to octal conversion

#59 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )

{
    int num1=-10;

    unsigned int num2=10;

    int ans= !!(num1<num2);

    printf("\n ans=%d", ans);

    return 0;
}
```

- **A**
ans = 0
- **B**
ans = 1
- **C**
ans = -10
- **D**
ans = 10

Correct Answer :A

Explanation

```

#include <stdio.h>

int main( void )
{

    int num1=-10;

    unsigned int num2=10;

    int ans= !!(num1<num2);

    // (num1<num2) => condition will return false

    //then we ! with false it will become TRUE

    //again we have one more ! the true become false

    //ans holds value 0 as integer datatype

    printf("\n ans=%d", ans);

    return 0;

}

```

while comparing `a<b` where `a` is `unsigned int` type and `b` is `int` type, `a` is type cast to `unsigned int` so,

On a typical implementation where `int` is 32-bit, -10 when converted to an `unsigned int` is 4,294,967,295 which is indeed <10 that result is false

What will be the output of the following C code?

```
#include <stdio.h>

int main(void)

{
    int i = 0;

    for (i = 0; i < 5; i++) {

        if (i <= 5)

        {
            printf("ccatpreparation \n");
        }
    }

    return 0;
}
```

- **A**
ccatpreparation print 5 times
- **B**
ccatpreparation print 4 times
- **C**
ccatpreparation print 1 times

- **D**
ccatpreparation print 0 times

Correct Answer :C

Explanation

```
#include <stdio.h>

int main(void)

{
    int i = 0;

    for (i = 0; i<5; i++);

        //here we have semicolon in the end so

        //loop will executed 5 times

    {

        //here we check are checking i < = 5 thats condition
        matches

        //output will be ccat

        if (i <= 5)
```

```
{  
  
    printf("ccatpreparation \n");  
  
}  
  
}  
  
return 0;  
  
}
```

#61 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>  
  
int main(void)  
  
{  
  
    int i = 0;  
  
    if (i == 0)  
  
    {  
  
        continue;  
  
        printf("ccatpreparation");  
    }  
}
```

```
    }

    return 0;

}
```

- **A**
ccatpreparation is printed infinite times
- **B**
ccatpreparation
- **C**
0
- **D**
Compile time error

Correct Answer :D

Explanation

error: continue statement cannot be used within a loop

continue statement is used inside a loop body to jump to the next iteration.

Here, you are using the continue statement inside the body of an if statement

and there is no loop around. You have to use loops (for, while & do-while)

in order to use continue.

#62 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )

{
    int k = 0;

    for ( ; k < 3; k++)

        printf("%d] ccatpreparation\t", k++);

    return 0;
}
```

- **A**
compile time error
- **B**
1] ccatpreparation
- **C**
0] ccatpreparation 1] ccatpreparation 2] ccatpreparation
- **D**
0] ccatpreparation 2] ccatpreparation

Correct Answer :D

Explanation

```
#include <stdio.h>

int main( void )
```

```
{  
  
    int k = 0;  
  
    for (;k < 3; k++)  
  
        printf("%d] ccatpreparation\t", k++);  
  
  
  
    //First time k initiaze with 0 then we check k < 3 then it  
    became TRUE  
  
    //then goes to print line and print k as 0 till k is not  
    incremented  
  
    //prints 0] pune  
  
    //now k++(post increment) inside print executed and k  
    become 1 then goes loop again  
  
    //where we have k++ againg it incremented by 1 now check k  
    < 3 then this  
  
    //condition become true again and it will print 2] pune  
  
    //now value of k become 4 loop condition will not satisfy  
    it exit the loop  
  
    return 0;  
  
}
```

What will be the output of the following C code?

```
#include <stdio.h>

int main(void)

{
    int i = 0, j=0, k=0;

    for(i=0, j=-3; i<3, j<3; i++, j++)

        k=i+j;

    printf("%d", i+j);

    return 0;
}
```

- **A**
12
- **B**
9
- **C**
11
- **D**
compile time error

Correct Answer :B

Explanation

```
int main(void)
```

```
{\n\n    int i = 0,j=0,k=0;\n\n    for(i=0,j=-3;i<3,j<3;i++,j++)\n\n        k=i+j;\n\n    printf("%d",i+j);\n\n    //First time i = 0 and j = -3 and k has value - 3\n\n    //second time =1 and j =- 2 and k has value 1 + -2 = -1\n\n    //third time =2 and j =- 1 and k has value 2 + -1= 1\n\n    //Fourth time =3 and j =0 and k has value 3 + 0 = 3\n\n    //fifth time =4 and j =1 and k has value 4 + 1 = 5\n\n    //Sixth time =5 and j =2 and k has value 5 + 2 = 7\n\n    // after loop complete i has value as 6 and j has 3\n    second\n\n    //final output will be 6 + 3 = 9\n\n    return 0;\n}\n
```

#64 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )

{
    int x = 100;

    if(x < 4);printf("%5d", x++);

    return 0;
}
```

- A
100
- B
101
- C
print nothing
- D
compile time error

Correct Answer :A

Explanation

```
#include <stdio.h>

int main( void )
```

```

{

    int x = 100;

    if(x < 4);printf("%5d", x++);

        //In above statement we have semicolon after if
        condition

        //so printf is not dependdent on if condition it will

        //execute directly but x++(post increment) so first
        value will print

        // then it incremented by 1

    return 0;

}

```

#65 Explained Report Bookmark

What will be the output of the following C code?

```

#include <stdio.h>

int main(void)

{

    int k;for (k = -3; k < -5; k++)printf("welcome ");

    printf("welcome ");

```

```
    return 0;  
}  
  
• A  
  Compile time error
```

- B
 welcome welcome
- C
 welcome welcome welcome
- D
 welcome

Correct Answer :D

Explanation

```
#include <stdio.h>  
  
int main(void)  
  
{  
  
    int k;  
  
    for (k = -3; k < -5; k++) printf("welcome ");  
  
    //in for loop second condition is not satisfy  
  
    // -3 < -5 it become false so for loop never executed  
  
    //welcome will print only 1 time  
  
    printf("welcome ");
```

```
    return 0;  
  
}
```

#66 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>  
  
int main(void)  
{  
  
    int val = -5 ;  
  
    while(++val < 5)  
        printf(" ccatpreparation",val++);  
  
    printf(" ",++val );  
  
    return 0;  
}
```

- **A**
ccatpreparation print Only 3 time
- **B**
ccatpreparation print Only 10 time
- **C**
ccatpreparation print Only 5 time
- **D**
ccatpreparation print Only 6 time

Correct Answer :C

Explanation

```
#include <stdio.h>

int main(void)

{
    int val = -5 ;

    //pre increment ++val so -4 < 5 condition true

    while(++val < 5)

        //after condition match it will print ccatpreparation

        //then increment value of val++(post increment)

        //ccatpreparation will print by 5 times

        //while(-4 <5)

        //ccatpreparation will print then val++ now value become -3

        //while(-2 <5) ++val incremented by 1 now value become -2

        //ccatpreparation will print then val++ now value become -1

        //while(0 < 5) ++val incremented by 1 now value become 0
```

```

//ccatpreparation will print then val++ now value become 1

//while(2 < 5) ++val incremented by 1 now value become 2

//ccatpreparation will print then val++ now value become 3

//while(4 < 5) ++val incremented by 1 now value become 4

//ccatpreparation will print then val++ now value become 5

//while(5 <- 5) condition false loop will exit

printf(" ccatpreparation",val++);

}

printf(" ",++val );

return 0;

}

```

#67[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include <stdio.h>

int main(void)

{

    int val = 0 ;

```

```

do

{

printf(" do while");

}while(val++);

val--;

while(val--);

printf(" while");

return 0;

}

```

- **A**
do while
- **B**
complie time error
- **C**
do while printed then infinite loop
- **D**
do while while

Correct Answer :D

Explanation

```

#include <stdio.h>

int main(void)

```

```
{  
  
    int val = 0 ;  
  
    do  
  
    {  
  
        printf(" do while");  
  
    }while(val++);  
  
    //in above loop condition first check with val value  
    that is 0 and  
  
    //condition consider as false so it will executed one  
    time only  
  
    //then val++ is incremented by 1 so val become val = 1  
  
    val--;  
  
    //in above val-- is post decrement before loop value  
    become 0  
  
    //while(0) will check but while loop has semicolon at  
    the end  
  
    //so below print statement is not dependent on while it  
  
    //simply executed  
  
    while(val--);
```

```
printf(" while");

//so final output will be do while while

return 0;

}
```

#68 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )

{

    int a=10;

    switch(0)

        case 0*5:

            printf(" case 0 %5d",a);

    switch(1)

        printf(" case 1 %5d",a);

return 0;
```

}

- **A**
no output
- **B**
case 0 10 case 1 10
- **C**
case 0 10
- **D**
compile time error

Correct Answer :C

Explanation

The default statement is executed if no case constant-expression value is equal to the value of expression . If there's no default statement, and no case match is found, none of the statements in the switch body get executed.

```
#include <stdio.h>

int main( void )
{
    int a=10;

    switch(0)

        case 0*5: // this expression evaluates as 0 * 5 = 0

            //here we are passing case as 0 which satisfy the
            condition
```

```
//below print statement will executed

    printf(" case 0 %5d",a);

switch(1)

    printf(" case 1 %5d",a);

// in above switch no case is match

//printf will not executed

// so final output will be case 0 10

return 0;

}
```

#69[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int main(void)
```

```
{
```

```
    int a=1;
```

```

switch(a++)

{
    default: printf("error");

    case 2 : case 1 : break;printf("ccat");

    case 3: printf("preparation");break;

}

return 0;
}

```

- **A**
testworld
- **B**
world
- **C**
test
- **D**
no output

Correct Answer :D

Explanation

```

#include <stdio.h>

int main(void)

{

```

```
int a=1;

switch(a++)

{

    // switch input a++ will be 2

    default: printf("error");

    case 2:// this cases matches the input paramater

    case 1 :

        break; // break keyword exit the seitch statement

        //without executing the printf statement

        printf("ccat");

    case 3:

        printf("preparation");

        break;

}

//So final output will blank

return 0;

}
```

#70 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main( void )
{
    int a = 0;
    while (++a++);
    {
        printf("welcome");
    }
    return 0;
}
```

- A
5
- B
6
- C
0
- D
compilation error

Correct Answer :D

Explanation

We can't use preincrement and postincrement operator same time on same variable.

#71 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main(void)

{
    int x;

    x=5>8 ? 10:1 !=5 <=5 ? 20 : 30;

    printf("Value of x:%d",x);

    return 0;
}
```

- **A**
Value of x:30
- **B**
Value of x:20
- **C**
Value of x:10
- **D**
Value of x:1

Correct Answer :A

Explanation

```
#include <stdio.h>

int main(void)

{
    int x;

    x=5>8 ? 10:1 !=5 <=5 ? 20 : 30;

    //first check 5 > 8 : condition will false it will jump to
    1!=5 <=5

    //in second condition first evaluate 5<=5 thats will return
    true (1)

    //then check 1 != 1 thats returns false , as per ternary
    operator

    //condition ? true : false , based on condition it will
    return value

    //so x is assigned to 30 value

    printf("Value of x:%d",x);

    return 0;
}
```

#72 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main(void)

{
    int a=33;

    do

    {
        printf("%d ", ++a);

        a-=3;

    }while(!!!(a <= 30+1));

    return 0;
}
```

- **A**
34
- **B**
33
- **C**
33 infinte loop
- **D**
34 infinte loop

Correct Answer :A

Explanation

```
#include <stdio.h>

int main(void)

{
    int a=33;

    do

    {
        printf("%d ", ++a);

        //++a pre-increment so a has 34 as updated value

        a-=3;

        //now a has 34 - 3 = 31 as updated value

    }while(!!!(a <= 30+1));

    //in while loop condition

    // (a <= 30+1) return TRUE

    // ! (a <= 30+1) true become FALSE
```

```
//!!(a <= 30+1) FALSE become TRUE  
  
//!!!(a <= 30+1) now in last TRUE become FALSE  
  
//so in while(false) : loop will exit  
  
//final output will be 34 only  
  
return 0;  
  
}
```

#73Explained Report Bookmark

What will be the output of the following C code?

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

```
int main(void)
```

```
{
```

```
    int val=1;
```

```
    switch(val)
```

```
{
```

```
    case 0+1+2*0:
```

```
        printf("ccat");
```

```
        if(++val%2==1);
```

```
else

{

break;

}

default: printf(" IT Park\n"); break;

}

return 0;

}
```

- **A**
ccat
- **B**
ccat exam
- **C**
compile time error
- **D**
none

Correct Answer :A

Explanation

```
#include <stdio.h>

int main(void)

{
```

```
int val=1;

switch(val)

{

    //input parameter to switch statement is 1

    case 0+1+2*0: // expression evaluate as 2 * 0 = 0

        //0 + 1 + 0 = 1 // this case will match the input para'

        printf("ccat"); // printf will be executed

        if(++val%2==1); //this condition will return false else
block will executed

    else

    {

        break; // this statement will break the switch

    }

default: //default case will not be executed

//because case 1 is already satisfy

printf(" IT Park\n");

break;
```

```
    }

    return 0;

//final output will be ccat

}
```

#74 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main(void)

{

    if (printf("welcome to ") == 0)

        if ((printf(" ccatpreparation ") + 1) % 2 == 0)

            printf(" online ");

        else

            printf(" offline ");

    return 0;

}
```

- A
welcome to ccat offline

- **B**
welcome to ccat online
- **C**
welcome to ccat
- **D**
compile time error

Correct Answer :D

Explanation

```
#include <stdio.h>

int main(void)

{
    if (printf("welcome to ")); //semicolon is not allowed in if condition
        //it will throw an compilation error
    if ((printf(" ccatpreparation ")+1)%2==0)
        printf(" online ");
    else
        printf(" offline ");
    return 0;
}
```

}

#75 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int function(int,int);

int main(void)

{

    int i=135,a=135,k;

    k=function(!++i,!++a);

    printf("i=%d a=%d k=%d\n",i,a,k);

    return 0;

}

int function(int j,int b)

{

    int c;

    c=j++ + b++;

    return !c;
```

}

- **A**
i=136 a=136 k=1
- **B**
i=136 a=136 k=0
- **C**
i=136 a=136 k=272
- **D**
i=135 a=136 k=1

Correct Answer :A

Explanation

```
#include<stdio.h>

int function(int,int);

int main(void)

{

    int i=135,a=135,k;

    //here we have ! sign in function call

    //so it will just pass 0,0 as value

    k=function(!++i,!++a);

    //return value from function is 1 so k = 1

    printf("i=%d a=%d k=%d\n",i,a,k);
```

```

//in function call we update function(!++i,!++a)

//now i = 136 and a = 136

//final outout will return i = 136 a = 136 k =1

return 0;

}

int function(int j,int b)

{

    int c;

    c=j++ + b++;

    //here we have post increment so

    //0 + 0 = 0 value in c

    return !c;

}

//in return statement again we have ! sign it will

//convert 0 to 1 so this function will return 1

```

#76 Explained Report Bookmark

What will be the output of the following C code?

```

#include<stdio.h>

int main(void)

{
    int i=10,j=20;

    printf("before fun call :: i=%d j=%d \n",i,j);

    i=i+j;

    j=i-j;

    i=i+j;

    j=i-j;

    i=i-j;

    printf(" after fun call :: i=%d j=%d \n",i,j);

    return 0;
}

```

- **A**
 before fun call :: i=10 j=20
 after fun call :: i=10 j=20
- **B**
 before fun call :: i=10 j=20
 after fun call :: i=20 j=10

- **C**
before fun call :: i=20 j=10
after fun call :: i=10 j=20
- **D**
compile time error

Correct Answer :A

Explanation

```
#include<stdio.h>

int main(void)

{
    int i=10,j=20;

    //simply print current value of i and j i =10 and j= 20

    printf("before fun call :: i=%d j=%d \n",i,j);

    i=i+j;//i =10 + 20 = 30

    j=i-j;//j =30 - 20 = 10

    i=i-j;//i =30 - 10 = 20

    i=i+j;//i =20 + 10 = 30

    j=i-j;//j =30 - 10 = 20

    i=i-j;//i =30 - 20 = 10
```

```
//final output will be i = 10 and j =20

return 0;

}
```

#77 Explained Report Bookmark

What will be the output of the following C code?

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

```
int main(void)
```

```
{
```

```
    int result;
```

```
    int i=10,j=20;
```

```
    result=add(i,j);
```

```
    printf("i=%d \n",result);
```

```
    return 0;
```

```
}
```

```
int add(int a,int b)
```

```
{
```

```
    int result;
```

```
result=a+b;

return result, a+b, a-b;

}
```

- **A**
Compile time error :function declaration is missing.
- **B**
 $i=40$
- **C**
 $i=30$
- **D**
 $i=-10$

Correct Answer :D

Explanation

```
#include<stdio.h>

int main(void)

{

    int result;

    int i=10,j=20;

    result=add(i,j); // here i =10 and j =20 pass as parameter

    //result will hold -10 return from function

    printf("i=%d \n",result);
```

```

//final output will be - 10

return 0;

}

int add(int a,int b)

{

    int result;

    result=a+b; // 10 +20 = 30

    return result, a+b, a-b;

//here inside return right most value will be return

//so 10 - 20 = -10 will return as output

}

```

#78[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include<stdio.h>

int myFunction(int,int);

int main(void)

{

```

```

int result;

int i=2,j=3;

i=myFunction(i,j);

printf("i=%d j=%d\n",i,j);

return 0;

}

int myFunction(int a,int b)

{

a=a+a;

b=b+b;

return b-b;

return a-a;

}

```

- **A**
i=4 j=6
- **B**
i=0 j=3
- **C**
i=4 j=0
- **D**
i=0 j=0

Correct Answer :B

Explanation

```
#include<stdio.h>

int myFunction(int,int);

int main(void)

{

    int result;

    int i=2,j=3;

    //input parameter as 2 and 3 to myFunction

    i=myFunction(i,j);

    //i holds 0 as return value

    printf("i=%d j=%d\n",i,j);

    //final output will be i = 0 and j = 3

    return 0;

}

int myFunction(int a,int b)
```

```
{  
  
    a=a+a; //a = 2 + 2 = 4  
  
    b=b+b; //b = 3 + 3 = 6  
  
    return b-b; //this return statement will return  
                //data as 6 - 6 = 0  
  
    return a-a; //this will not executed  
  
    //also there is no error for multiple return  
  
    //only first return will executed  
  
}
```

#79[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>  
  
int function(int z);  
  
int main(void)  
  
{  
  
    int z=111;  
  
    z = z + function(z++);
```

```
    printf("result=%d", z);
```

```
    return 0;
```

```
}
```

```
int function(int z)
```

```
{
```

```
    return ++z;
```

```
}
```

- **A**
result=225
- **B**
result=224
- **C**
result=222
- **D**
result=223

Correct Answer :B

Explanation

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

```
int function(int z);
```

```
int main(void)
```

```
{
```

```
int z=111;

//input parameter to function is 111 z++ is post increment

// first pass the value then increment will happen

z = z + function(z++);

//so z increment by 1 updated value will be 112

// z = z + function(z++);

// z = 112+ 112 =224

printf("result=%d", z);

//final output will be 224

return 0;

}

int function(int z)

{

//before return z =111 incremented by 1

//++z isa pre increment so 112 will be return

return ++z;

}
```

#80 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int main(void)

{

    if((printf("Hello C\n")-8))

    {

        main();

    }

    printf("Hello C\n");

    return 0;

}
```

- A stack overflow error
- B prints Hello C only once
- C prints Hello C twice
- D prints Hello C infinite number of times

Correct Answer :C

Explanation

```
#include<stdio.h>

int main(void)

{
    if((printf("Hello C\n")-8))

        // (printf("Hello C\n") will executed

        // and return numbers of characters successfully printed

        // 8 -8 =0 so if condition will false

        // it will not go inside the block statement

    }

    main();

}

printf("Hello C\n"); // simple Hello C will print

return 0;

}
```

#81 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

void fun(int);

int main( void )

{

    static int i=1; fun(i);

    return 0;

}

void fun(int i)

{

    static int j=1;

    i=j;j++;i++;

    printf("%d,",i);

    if(i<=3)

        fun(i);

}
```

- **A**
1,1,1
- **B**
2,2,3

- **C**
2,3,4
- **D**
3,4,5

Correct Answer :C

Explanation

```
#include<stdio.h>

void fun(int);

int main( void )

{

    //static variable holds the value in data segment area

    static int i=1; // i declared as static and initialize i =
1

    fun(i);//pass 1 as input

    return 0;

}

void fun(int i)

{

    static int j=1;//j declared as static with value 1
```

```

i=j;

j++; //j will incremented by 1

i++; //i will incremented by 1

printf("%d", i);

//first time it will print 2 i++ incremented by 1

//second time it will print 3 i++ incremented by 1

//because value of i is preserved (static variable)

//third time it will print 4 i++ incremented by 1

//because value of i is preserved (static variable)

if(i<=3)

fun(i);

}

```

#82 Explained Report Bookmark

What will be the output of the following C code?

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

```
void rec(int);
```

```
int main()
{
    int a=3;
    rec(a);
    return 0;
}

void rec(int n)
{
    if(n>0)
    {
        rec(--n);
        printf(", %d", n);
        rec(--n);
    }
    else
        printf("\t");
}
```

- **A**
,1,2,0,0
- **B**
,0,1,2,1
- **C**
,0,1,2,0
- **D**
,0,1,0,0

Correct Answer :C

Explanation

In this program, recursive function calls are made. There are two ways to solve this type of problem manually. One is with the help of stack and other way is with the help of recursion tree.

#83 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

int main(void)

{
    int i;

    for(i=0;i<3;i++)
    {
        int x=0;
```

```

static int y=0;

printf("x=%d , y=%d \t",x++,y++) ;

}

return 0;
}

```

- **A**
x=0,y=0 x=1,y=1 x=2,y=2
- **B**
x=0,y=0 x=1,y=0 x=2,y=0
- **C**
x=0,y=0 x=0,y=1 x=0,y=2
- **D**
x=0,y=0 x=0,y=0 x=0,y=0

Correct Answer :C

Explanation

```

#include<stdio.h>

int main(void)

{
    int i;

    for(i=0;i<3;i++)

{

```

```
    int x=0;// block level x Initialization value will
persist within {}

    static int y=0; // value will preserved on data stack for
each iteration

printf("x=%d , y=%d \t",x++,y++);

//first time x =0 and y =0 will print

//because x++ and y++ are post increment first value will
print then

//increment will happen

//similar for second time

//x = 0 but y =1 because declared as static

//second time it will print x =0 and y =1 then y is
incremented by 1

//similar for third time

//x = 0 but y =2

}

//so final output will be x=0 , y =0      x=0 , y =1      x=0 ,
y =2

return 0;

}
```

#84 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

register int i;

int main(void)

{

    printf("\n Enter value of i:::");

    scanf("%d",&i);

    printf("\n i=%d i=%u", i, &i);

    return 0;

}
```

- **A**
register variables can not declare globally
- **B**
we can not print the address of register variables
- **C**
Both A and B
- **D**
Run time error

Correct Answer :C

Explanation

Point 1 :

Registers can only be reserved for variables of pointer type. A global register variable cannot be initialized. The register dedicated for a global register variable should not be a volatile register, or the value stored into the global variable might not be preserved across a function call.

point 2 :

You can't access the address of a register variable because there is no such thing. Memory addresses describe memory locations. They do not describe CPU registers, I/O ports, TLB entries, or whatever else can hold data.

The C language standard has a rule for what can be used with the address-of operator, paragraph 6.5.3.2 (emphasis mine)

The operand of the unary & operator shall be either a function designator, the result of a [] or unary * operator, or an lvalue that designates an object that is not a bit-field and is not declared with the register storage-class specifier.

#85 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

void func(void);

int main(void)
```

```

{

    func(); func();

    return 0;

}

void func(void)

{

    auto int i=0;

    register int j=0;

    static int k=0;

    i++; j++; k++;

    printf("i=%d j=%d k=%d\t", i, j, k);

}

```

- **A**
i=1 j=1 k=1 i=1 j=1 k=2
- **B**
i=0 j=0 k=0 i=0 j=0 k=0
- **C**
i=1 j=1 k=1 i=2 j=2 k=2
- **D**
compile time error

Correct Answer :A

Explanation

auto and register variables take memory and free memory as function in which they reside is called and finished its execution. But this is not the case for static variable because the life of the static variable is throughout the program but not the life of function. Static variables are not destroyed after function execution is finished.

#86 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main(void)

{
    extern int var=1000;

    printf(" var = %d", ++var);

    return 0;
}

• A
    var = 1000
• B
    var = 0
• C
    var = 1001
• D
    compile time error
```

Correct Answer :D

Explanation

: Compiler Error will be like this

[Error] 'var' has both 'extern' and initializer.

#87 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int fun(float a);
```

```
int main( void )
```

```
{
```

```
    static float x;
```

```
    x=(float)fun(100);
```

```
    printf(" x = %.f ",x);
```

```
    return 0;
```

```
}
```

```
int fun(float a)
```

```
{  
  
    return a ==100.0f ? 1000 : 500;  
  
}
```

- **A**
compile time error
- **B**
 $x = 1000.000000$
- **C**
 $x = 500$
- **D**
 $x = 1000$

Correct Answer :D

Explanation

.f in c? %f is the format specifier used for a float data type in the functions printf and scanf .

By default it will display values up to 6 digits after the decimal point but, writing %.1f or %.2f will reduce the precision to 1 or 2 digits respectively.

when you use % .f that mans no decimal

#88 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
static int num=9;

int main(void)

{

    if (num<0)

        return 0;

    else if (num%2==1)

    {

        num--;

        printf("%3d,", num-=1);

    }

    else

    {

        printf("%3d,", num-=2);

    }

main();

return 0;

}
```

- **A**
8, 6, 4, 2, 0,
- **B**
8, 5, 4, 1, 0, -3,
- **C**
8, 6, 4, 2, 0, -2,
- **D**
7, 5, 3, 1, -1,

Correct Answer :D

Explanation

num is declared as a static variable it means its value will be preserved throughout the program.

num<0 condition false control moves to else if block

num%2 = 9%2 = 1 (% returns remainder on dividing 9 by 2)

Hence num%2 == 1 true

Now num-- decrement num by 1 [num=8]

In the printf statement num is again decremented by 1

(num-=1 it is a shorthand method of writing num = num-1) [num=7]

Hence 7 is printed.

NOW main() is called recursively however this time value of num is 7.

Similar procedures will be repeated and 5, 3, 1, -1 will be printed.

The value of num is -1.

Now when main() is called recursively after printing -1 the if(num<0) condition becomes true and it

will return 0 .

Hence the output is

7 5 3 1 -1

#89 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

int fun(int x,int y)

{
    if (x==0)
        return y;
    return fun(x-1,x+y);
}
```

- A
X is 5

- **B**
X is 3
- **C**
X is 2
- **D**
Compile time error

Correct Answer :D

Explanation

Compiler error will be like this

undefined reference to `WinMain'.

#90 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

signed char* convert(unsigned int *data)

{
    return ((signed char*) (data));
}

int main( void )
{
}
```

```
unsigned int a = 577;  
  
signed char *data=NULL;  
  
data =convert(&a);  
  
printf(" %d ",*data);  
  
return 0;  
  
}
```

- **A**
66
- **B**
B
- **C**
A
- **D**
65

Correct Answer :D

Explanation

Range of unsigned int = 0 to 65,535 or 0 to 4,294,967,295.

Range of signed char = -127 to 128 (total 256 numbers are allowed)

address of variable a (unsigned int) is passed in convert() and signed version of variable a is assigned to pointer data.

Now since data is of type character pointer and that range of signed character is between -127 to 128 therefore we need to convert 577 into allowable range.

hence $577 \% 256 = 65$ (A) is assigned to pointer data.

// modulo operator returns remainder.

%d is used to print the final output hence 65 is printed.

#91 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

void calculate(float *ptr_floater1, float *ptr_floater2,
float *ptr_answer);

int main( void )

{
    float floater1 = 0.001, floater2= 0.003, answer=0.0f;
    calculate(&floater1,&floater2, &answer);
    printf("ans=% .3f", answer);

    return 0;
}
```

```

void calculate(float *ptr_floater1, float *ptr_floater2,
float *ptr_answer)

{
    *ptr_floater1 = 4 * *ptr_floater1;

    *ptr_answer= 3 * (*ptr_floater2 - *ptr_floater1);

    return;
}

```

- **A**
0.001
- **B**
-0.003
- **C**
0.004
- **D**
0.003

Correct Answer :B

Explanation

$\text{floater1} = 0.001$, $\text{floater2} = 0.003$, $\text{answer} = 0.0$

Now, these variables are passed by address in the function calculation.

Therefore *ptr_floater1 points to variable floater1 similarly *ptr_floater2 points to variable floater2 and *ptr_answer points to variable answer .

Therefore the changes made by the pointers will reflect in variables value.

$$*\text{ptr_floater1} = 4 * *\text{ptr_floater1}$$

$$= 4 * (\text{value at address pointed by ptr_floater1})$$

$$= 4 * (0.001)$$

$$= 0.004$$

$$*\text{ptr_answer} = 3 * (*\text{ptr_floater2} - *\text{ptr_floater1})$$

$$= 3 * (\text{value at address pointed by ptr_floater2} - \text{value at address pointed by ptr_floater1})$$

$$= 3 * (0.003 - 0.004)$$

$$= -0.003$$

Now the control returns and the printf statement after calculate () function is executed which prints the value stored in variable answer.

Variable answer currently store -0.003 because its value is changed by *ptr_answer inside calculate() function.

What will be the output of the following C code?

```
#include <stdio.h>

int x=10;

void callbyaddress(int *ptr_x)

{

    x=*ptr_x * *ptr_x / x;

}

int main( void )

{

    int x=100;

    printf(" x = %d ",x);

    callbyaddress(&x);

    printf(" x = %d",x);

    return 0;

}
```

- **A**
x = 100 x = 100
- **B**
x = 100 x = 10

- **C**
 $x = 10$ $x = 10$
- **D**
Compile time error

Correct Answer :A

Explanation

`int x=10;` is a global variable initialized outside `main()` function.

`int x=100;` is a local variable initialized inside a `main()` function.

Inside a function local variable is given higher priority than the global variable of the same name.

Inside `main()` function the first `printf` statement will print `x=100` since the local variable `x` is given higher priority inside the `main` function.

Now the function `callbyaddress(&x)` is called the address of LOCAL VARIABLE `x` is stored in pointer `ptr_x`;

Now

`x= *ptr_x * *ptr_x / x` is performed

remember here `x` variable is the global variable `x` while `ptr_x` stores the address of the local variable `x` of `main()` function.

Precedence of division is greater than multiplication

$x = 100 * 100/10 = 1000$

this 1000 is stored in global variable x;

Now the control returns to the main() function and encounter the second printf statement and it print the value of x but watch out, here x variable is still the local variable which is still storing the value 100. (local variable enjoys higher priority inside a function)

The function callbyaddress() changes the value of global variable x while pointer ptr_x does not alter the value stored at the address indicated by it.

Hence the output will be

x=100 x=100

#93 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int x=10;

void callbyaddress(int *ptr_x)

{
    x=*ptr_x * *ptr_x / x;

}

int main( void )
```

```
{  
  
    int x=100;  
  
    printf(" x = %d ",x);  
  
    callbyaddress(&x);  
  
    printf(" x = %d",x);  
  
    return 0;  
  
}
```

- **A**
x = 100 x = 100
- **B**
x = 100 x = 10
- **C**
x = 10 x = 10
- **D**
Compile time error

Correct Answer :A

Explanation

The key concept here is that local variable is given higher preference in the same block.

global, variable x =10

local to main function ,variable x = 100

In the main function value of x=100 so the first printf function will print 100 since local variable has higher preference.

Now in the function

```
void callbyaddress(int *ptr_x)  
{  
    x= *ptr_x *ptr_x/ x  
}
```

here *ptr_x is 100 since address of variable x(local to main)is passed to function.

and here x is 10 since in callbyaddress function global version of variable x is used which is 10

therefore $x=100*100/10 =1000$ hence value of (global) variable x is 1000;

Now the control returns to main() function

here it encounter the printf function again but still see that the value of x is again printed using the local variable x whose value is still 100;

hence the final output will be x=100 x=100 since both the times local variable x is used.

#94 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

void callbyaddress(float *ptr_val)
{
    *ptr_val= 2.0f * *ptr_val;

    return;
}

int main( void )
{
    float val=3.14f;

    float * const strVal = &val;

    callbyaddress(&val);

    printf("%.1f", *strVal);

    return 0;
}
```

- A
3.1
- B
6.3
- C
6.2
- D
compile time error

Correct Answer :B

Explanation

Program Explanation

```
#include<stdio.h>

void callbyaddress(float *ptr_val)

// (step 3)

// *ptr_val store the address of variable 'val' which holds
the value 3.14

{

*ptr_val= 2.0f * *ptr_val;
```

```
// (step 4)

// *ptr_val = 2.0 * value at the address pointed by the
pointer *ptr_val

// *ptr_val = 2.0 * 3.14 = 6.28

return;

}

int main( void )

{

float val=3.14f;

float * const strVal = &val;
```

```
// (step 1)          the above line means the address stored in
'strVal' cannot be changed //                                but the
value stored in that address can be changed.
```

```
//                      strVal store the address of the
variable 'val'
```

```
callbyaddress(&val);
```

```
// (step 2)          the address of variable 'val'
is passed in the function.
```

```
printf("%.1f", *strVal);
```

```
// (step 5)
```

```
//           this line print the value store in the address  
pointed by the pointer strVal  
  
//           do note that %.1f is used, not %f  
  
//           " %.1f " is used to print float value with only  
one digit after the dot however it  
  
//           also round-figure the value therefore 6.28 changes  
to 6.3 (6.22 changes to  
  
//           6.2)  
  
return 0;  
}
```

#95 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

void modify(int * const value)

{
    *value = 200;

    return ;
}

int main( void )

{
    const int value = 300;

    modify(&value);

    printf("value = %d\n", value);

    return 0;
}
```

- **A**
value = 200
- **B**
value = 300
- **C**
garbage value
- **D**
none of above

Correct Answer :A

Explanation

Const int value = 300

Here value is a constant variable its value cannot be changed throughout the program by simple means however there is an exception in it that value of a constant variable can be changed by using pointer. This concept is used in this problem.

Address of constant variable value is passed in function modify(&modify).

This address is collected by the pointer VALUE.

Now there is another concept, check this line

int * const value

Here VALUE is a constant pointer to the variable value.

The address pointed to by pointer VALUE cannot be changed however the data pointed by the pointer can be changed.

*value = 200

This will change the value of the variable value. Hence the printf statement will output

Value = 200

#96 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int no=1000;

int* fun1(int *value)

{

no += *value / *value;

return &no;

}

int main( void )

{

int num1=10;

int *val=fun1(&num1);

printf(" value= %d", *val);

return 0;

}
```

- A
value = 1001

- **B**
value = 101
- **C**
value = 1010
- **D**
run time error

Correct Answer :A

Explanation

num1 =10;

address of num1 is passed in the function fun1(&num1)

This address is collected in the pointer * value.

Now

no += *value / *value

Precedence of / is greater than += hence division is performed first

$$*value / *value = 10 / 10 = 1$$

Now, no+=1 (shorthand method of writing no = no+1)

no=no+1 = 1000+1 (no is a global variable whose value is 1000)

```
no =1001
```

Now address of global variable no is returned and it is stored in the pointer *val of the main() function.

Printf statement prints the value pointed by the pointer val

Hence output is

Value = 1001

#97 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )
{
    int num1 = 100;
    char ch='D';

    void *ptr_num1 = &num1;

    printf("%c-",--*(int*)ptr_num1);

    ptr_num1=&ch;

    printf("%c",--*(char*)ptr_num1);
```

```
    return 0;  
}
```

- A
100 - D
- B
99 - C
- C
c - C
- D
D - D

Correct Answer :C

Explanation

For solving this question we first need to clear our concepts about void pointer.

A void pointer is a pointer which can point to any data type but in order to use the value pointed to by void pointer we first need to typecast it into proper pointer type.

Here ptr_num1 is a void pointer which has assigned the address of a integer value.

So ptr_num1 is first type casted into integer pointer `(int*)ptr_num1`

Now it's value is accessed using * (dereferencing operator) `*(int *)ptr_num1`

Now the value pointed by ptr_num1 which is 100 is decremented by 1 --
`*(int *)ptr_num1`

So ptr_num1 points to integer value 99

However we have a print statement with %c that means the char value associated with Ascii 99 is printed which is 'c'

The same process goes for ptr_num1 = &ch

Here void pointer points to char variable that means we first need to typecast it into char pointer (char*)ptr_num1

Now it's value is accessed using *(char *)ptr_num1

Hence ptr_num1 points to 'D'

Now it is decremented using --

--*(char*)ptr_num1

That means ptr_num1 points to 'C' value

Also the print statement use %c so it will print 'C'

Hence the final output will be

c-C

#98 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```

int main( void )
{
    void *ptr=NULL;
    char ch=72;
    int no='i';
    float f=4.46;
    ptr=&ch; printf("%c", *(char*)ptr);
    ptr=&no; printf("%c", *(int*)ptr);
    ptr=&f; printf("-%.f", *(float*)ptr);
    return 0;
}

```

- **A**
Hi-5
- **B**
Hi-4
- **C**
Hi-4.460000
- **D**
Garbage value

Correct Answer :B

Explanation

Concept: A void is pointer that can point to any datatype but in order to use the value pointed to by void pointer we first need to typecast it into proper pointer type.

Here ptr is a void pointer.

First ptr is assigned the address of char ch. (ch = 72)

```
printf("%c", *(char*)ptr)
```

ptr is typecasted into char pointer and its value is accessed using *
(char*)ptr and printed using %c therefore the character associated with
ascii 72 is printed that is 'H'

Second ptr is assigned the address of interger no (no = 'i')

```
ptr=&no; printf("%c", *(int*)ptr);
```

ptr is typecasted into interger pointer and its value is accessed using *
(int*)ptr and printed using %c therefore the character 'i' is printed.

Third ptr is assigned the address of float f (f = 4.46)

```
ptr=&f; printf("-%.f", *(float*)ptr);
```

ptr is typecasted into float pointer and its value is accessed using
(float)ptr and printed using “-%.f”

Here character ‘ -‘ is printed as it is.

%.f will truncate all the digits after the decimal and prints only the whole number. That is only 4 is printed.

Hence the final output will be Hi-4

#99 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )
{
    int num1 = 100, num2=150;

    int *p = &num1, **pp = &p;

    **pp = num2;

    ++**pp;

    printf("num1=%d *p=%d **pp=%d\n", num1, *p, **pp);

    return 0;
}
```

- **A**
num1=101 *p=101 **pp=101
- **B**
num1=151 *p=151 **pp=151
- **C**
num1=101 *p=100 **pp= garbage value

- D
num1=151 *p=151 **pp= garbage value

Correct Answer :B

Explanation

B is the right answer. * means pointer to a variable and ** means pointer to a pointer which is pointer to some variable. &p is the address of a pointer. ++ will first increment the value of **pp value and pass the value of it i.e. it is a pre increment operator. This program is basically interchanging the values using addresses(pointers). Incrementing will increment the num2 which is 150 to 151 which will also be shared with num 1 since the pointer operation is performed

&num1: 100 &p: 6422016

*p: 100 **pp: 100

**pp: 150 num2: 150

++**pp: 151

num1=151 *p=151 **pp=151

#100 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )
```

```

{

    int num1 = 100, num2=150;

    int *p = &num1;

    int **pp = &p;

    **pp = 75;

    --*p;

    ++**pp;

    num2=**pp;

    printf("num1=%d num2=%d ", num1, num2 );

    printf("*p=%d **pp=%d\n", *p, **pp );

    return 0;
}

```

- **A**
num1=100 num2=150 *p=100 **pp=100
- **B**
num1=100 num2=75 *p=74 **pp=75
- **C**
num1=150 num2=150 *p=150 **pp=150
- **D**
num1=75 num2=75 *p=75 **pp=75

Correct Answer :D

Explanation

D is the right answer. * means pointer to a variable and ** means pointer to a pointer which is pointer to some variable. &p is the address of a pointer. ++ will first increment the value of **pp value and pass the value of it i.e. it is a pre increment operator. This program is basically. This program is reallocating a value to a variable. Changing the value of **pp will directly change the value of *p since they are pointing to the same location or value. And then by incrementing and decrementing by 1. You can notice the working of the program below.

*p: 100

&p: 6422028

&p: 6422016

**pp: 100

*p: 75

**pp: 75

*p: 74

--*p: 74

++**pp: 75

**p: 75

num1=75

num2=75

*p=75

**pp=75

#101 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int fun(int **pp);

int main( void )
{
    int val = 10 , *ptr1= &val;
    val=fun (&ptr1);
    printf(" val=%d *ptr=%d", val, *ptr1);

    return 0;
}

int fun(int **pp)
{
```

```

    **pp *= **pp ;

printf(" **pp=%d ", **pp);

return **pp/10;

}

```

- **A**
 **pp=100 val=10 *ptr=10
- **B**
 **pp=100 val=10 *ptr=100
- **C**
 **pp=100 val=100 *ptr=10
- **D**
 **pp=10 val=10 *ptr=10

Correct Answer :A

Explanation

A is the answer.

val: 10 *ptr1: 10 &val: 6422044 &ptr1: 6422032

Inside method process:

**pp: 10

print: **pp=100

**pp/10: 10

Outside method: val: 10

print: val=10 *ptr=10

As you can see, here pointer to pointer is used to point “val” variable. Then *= operator multiplies the left variable with right variable and stores it on left left variable which is **pp. Again the division operation is performed on **pp i.e. value 100 by performing **pp/10 operation which will return 10 value.

#102Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

void changeVal2(int **x)

{
    **x/=2;

    return;
}

void changeVal1(int *x)

{
    *x*=5;

    return;
}
```

```

}

int main( void )
{

    int num1=10;

    int *ptr1=&num1;

    changeVal2(&ptr1);

    printf("num1=%d *ptr1=%d ",num1, *ptr1);

    changeVal1(ptr1);

    printf("num1=%d *ptr1=%d ",num1, *ptr1);

    return 0;
}

```

- **A**
num1=10 *ptr1=5 num1=10 *ptr1=25
- **B**
Run time error
- **C**
num1=5 *ptr1=5 num1=25 *ptr1=25
- **D**
Compile time error

Correct Answer :C

Explanation

```
#include<stdio.h>

void changeVal2(int **x)

// (step 2)

// // x is a pointer to a pointer and store the address of
ptr1 is 9000

{

    **x/=2;

// (step 3)

// **x/=2      ->      **x = **x / 2

// ->      **x = **(9000)/2 =      *(1000)/2      =      10 /2      =5

// value at address 9000 is an another address 1000 and value
at address 1000 is 5.

// therefore **x =5  that means value in num1 is 5 and value
at the address pointed

// by the pointer ptr1 is also 5.

return;
```

```
}

void changeVal1(int *x)

// (step 6)

// x is a pointer which holds the address 1000.

{

    *x*=5;

// (step 7)

// *x*=5      ->      *x = *x * 5

->      *x = *(1000) * 5  (value at 1000 is 5)

->      *x = 5 * 5 = 25

// therefore *x =5  that means value in num1 is 25 and value
at the address pointed

// by the pointer ptr1 is also 25.

return;
```

```
}

int main( void )

{

// Suppose the address of variable 'num1' is 1000 , address of
pointer ptr1 is 9000

    int num1=10;

    int *ptr1=&num1;

// in above line address of num1 is stored in pointer ptr1

    changeVal2(&ptr1);

// (step 1)

// here address of ptr1 (i.e 9000) is passed as function
argument.

    printf("num1=%d *ptr1=%d ",num1, *ptr1);

// (step 4)

// it prints 5 5

    changeVal1(ptr1);

// (step 5)

// value stored in ptr1 i.e 1000 is passed in the function.
```

```
printf("num1=%d *ptr1=%d ", num1, *ptr1);

// (step 8)

// it prints 25 25

return 0;

}
```

#103 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

int main( void )

{

    const int a = 10;

    int * const ptr = &a;

    *ptr = a*a;

    printf("a = %d ptr = %d ", a, ++*ptr);

    printf("a = %d ptr = %d ", a, --*ptr);

    return 0;

}
```

- **A**
Compile time error
- **B**
Run time error
- **C**
 $a = 101$ $ptr = 101$ $a = 100$ $ptr = 100$
- **D**
 $a = 100$ $ptr = 101$ $a = 100$ $ptr = 99$

Correct Answer :C

Explanation

```
#include<stdio.h>

int main( void )
{
    const int a = 10;

    // a is a constant variable that means its value
    cannot be changed however it can be

    // changed by means of pointer,

    int * const ptr = &a;

    // ptr is a constant pointer which store the
    address of variable a.
```

```
*ptr = a*a;

// *ptr = a*a = 10 * 10 = 100

// *ptr = 100 ( i.e value at the address stored in ptr is 100)

printf("a = %d ptr = %d ", a,++*ptr);

// here first ++*ptr is evaluated and printed then secondly a
is printed.

// precedence of ++ > * so ++ is evaluated first

// ++(*ptr) = *ptr = *ptr + 1

// = *ptr = 100 +1

// = *ptr = 101

// since ptr is a pointer, therefore, change in value also
occurred at the variable

// 'a' hence value store in a is also 101.

// finally 101 101 is printed using the above the line of
code.

printf("a = %d ptr = %d ", a,--*ptr);

// here first --*ptr is evaluated and printed then secondly a
is printed.
```

```

// precedence of -- > * so -- is evaluated first

// --(*ptr) = *ptr = *ptr - 1

//           =      *ptr = 101 -1

//           =      *ptr = 100

// since ptr is a pointer, therefore, change in value also
// occurred at the variable

// 'a' hence value store in a is also 100.

// finally 100 100 is printed using the above the line of
// code.

// final output : a=101 ptr=101 a=100 ptr=100

return 0;

}

```

#104 Explained Report Bookmark

What will be the output of the following C code?

```

#include<stdio.h>

int main( void )

```

```

{

const int a = 10 , b = 20 ;

const int * ptr = &a;

int * const ptr1 = &a;

printf("a = %d *ptr = %d ", a,*ptr);

ptr = &b;

*ptr1= b;

printf("a = %d *ptr = %d ", a,*ptr);

return 0;
}

```

- **A**
a = 10 *ptr = 10 a = 10 *ptr = 20
- **B**
a = 10 *ptr = 10 a = 20 *ptr = 20
- **C**
compile time error
- **D**
run time error

Correct Answer :B

Explanation

&a: 6422028

*ptr: 10

ptr1: 6422028

Print a = 10 *ptr = 10

&b: 6422024

ptr: 6422024

b: 20

*ptr1: 20

Print a = 20 *ptr = 20

As you can see, only basic operations are performed in this program. As you can see const keyword is used here in order to maintain the address of ptr. But since the value of b is shared with variable a through pointer, at the end of the program they have same values which is 20.

#105 [Not Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>

int fun1(int n1, int n2)

{
    return n1+n2;
}
```

```
}

int fun2(int n2, int n1)

{

    return n1-n2;

}

void print(int n1, int n2,int (*fp)(int n1, int n2) )

{

printf(" funptr = %d (*funptr) = %d" , fp, *fp);

    return;

}

int main( void )

{

int val1=100, val2=200;

int (*funptr)(int no1, int no2);

funptr=fun1(val1,val2);

print(val1, val2, funptr);

funptr=fun2(val2, val1);
```

```
print(10, 20, funptr);  
  
return 0;  
  
}
```

- **A**
run time error
- **B**
funptr=300 (*funptr)=300 funptr= 100 (*funptr)= 100
- **C**
funptr=300 (*funptr)=300 funptr=-100 (*funptr)=-100
- **D**
compile time error

Correct Answer :C

No Explanation Available

#106 [Explained](#) [Report](#) [Bookmark](#)

What is the meaning of the statement?

```
int* fun(char (*a) []);
```

- **A**
fun is a function that accepts an argument which is a pointer to a character array returns a pointer to an integer.
- **B**
fun is a function that accepts an argument which is an array of pointers to characters returns a pointer to an integer.

- **C**
fun is a function that accepts an argument which is a pointer to a character array returns an integer.
- **D**
fun is a function that accepts an argument which is an array of pointers to characters returns an integer.

Correct Answer :A

Explanation

The statement decrypted as per precedence table.

Precedence is as follows

- 1] () ----- function call
- 2] [] ----- Array subscripting
- 3] * ----- Indirection(dereference)

#107 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
void fun();
int main( void )
{
    fun();
    return 0;
}
```

```

void fun()
{
    #ifndef value
    #define value 100
    #undef value
    #else
    #undef value
    #define value 200
    #endif
    #define Value 300 printf("Value : %d",Value);
    return ;
}

}

```

- **A**
Value : 100
- **B**
Value : 200
- **C**
no output
- **D**
Value : 300

Correct Answer :C

Explanation

Here we are calling function fun from the main method.

1. **#ifndef value** :check if value not defined
2. **#define value 100** :if not then define it as 100
3. **#undef value** : undefine or remove data from value
4. **#else** :if value exist then follow next block
5. **#undef value**: undefine value
6. **#define value 200**: define value to 200

7. #endif: end if block , that means next block of code is free block which does not depend on any if statement mentioned above
8. #define Value 300 printf("Value : %d",Value); : define value to 300, but since the printf statement mentioned inside #define it wont get executed. If we write it on next line then it will print Value: 300.
9. return ; return the control to called line
10. As you can clearly see that no output will be prints on the screen because of the 8th note.

#108 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#line 100
int main( void )
{
    printf("\n line = %d ", __LINE__ );
    #line 0
    printf(" line = %d ", __LINE__ );
    #line 100
    printf(" line = %d ", __LINE__ );
    return 0;
}
```

- **A**
run time error
- **B**
compile time error
- **C**
line = 102 line = 0 line = 100
- **D**
line = 103 line = 0 line = 102

Correct Answer :C

Explanation

In c programming every instruction has a line number which is usually start with 1.

This line number can be modified using #line directive and the current line number can be accessed using the __LINE__

In the given program , #line 100 will define a line number 100 to the next line after it is written

That means ' { ' will be at line number 101

and the very first printf() function will be at line number 102. hence __LINE__ is currently set to 102.

Now again #line will set the next line number to 0 hence the next printf() function will print 0 since __LINE__ is currently set to 0.

similarly #line will set the next line number to 100 hence the last printf() function will print 101 since __LINE__ is currently set to 100

#109Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#include<string.h>
int main( void )
{
    #define CCAT "CCAT PREPARATION \n"
    #define ccat "CCAT PREPARATION ONLINE\n"
    #ifdef CCAT
    printf(CCAT );
}
```

```

#endif
#ifndef CCAT
printf(ccat );
#endif
printf("\'ccat \'\\n");
printf("\"CCAT \" \\n");
return 0;
}

```

- **A**
CCAT PREPARATION
CCAT PREPARATION ONLINE
'ccat '
"CCAT "
- **B**
CCAT PREPARATION
- **C**
CCAT PREPARATION
CCAT PREPARATION ONLINE
- **D**
CCAT PREPARATION ONLINE
'ccat '
"CCAT "

Correct Answer :A

Explanation

1. Here we print CCAT and ccat which is defined using #define which makes the variable or the operation open to program or local block depending on where it is defined.
2. As CCAT exists which is checked by #ifdef and ended by #endif holds “CCAT PREPARATION” and also as ccat exists which is checked by #ifdef and ended by #endif holds “CCAT PREPARATION ONLINE” its printed on screen

3. Then the printf statement prints ccat and CCAT where "\\" is still the same as "" - you have to escape the backslash.

#110 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#define A(x) ((x)*(x))
int main( void )
{
    int a, b=3;
    a = 75 / (b* A((b+2)));
    printf("%d\n", a);
    return 0;
}
```

- **A**
1
- **B**
625
- **C**
75
- **D**
225

Correct Answer :A

Explanation

b=3

a = 75 / (b* A((b+2)));

a= 75 / (3 * A ((3+2)))

a= 75 / (3 * (3+2) * (3+2))

a= 75 / (3 * (25))

a=1

#111 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>
#define MACRO(n, i, a, m) m##a##i##n
#define MAIN MACRO(n, i, a, m)
#define CCAT ccat
int MAIN(void )
{
    printf("\\"CCAT\\\"");
    return 0;
}
```

- **A**
"CCAT"
- **B**
CCAT
- **C**
ccat
- **D**
compile time error

Correct Answer :A

Explanation

- As MAIN is defined using #define which helps to define macros, the program first jump to #define MAIN MACRO(n, i, a, m)
- Then from there MACRO(n,i,a,m) is executed since it is also defined using #define
- Then, #define MACRO(n, i, a, m) m##a##i##n line is explored. Here ## perform concatenation operation which makes it “main”.
- Then “main” is returned to int MAIN(void) where it is recognised as int main(void)
- Here printf("\"CCAT\""); is explored. But the doubt can arise as CCAT is also defined as macro. But since we are using \ it is ignored and "CCAT" is printed.

#112 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#define SWAP(a, b) {b ^= b; a ^= a; b ^= b ;}
int main( void )
{
    int x = 10;
    int y = 20;
    x=x*y; y=x/y; x=x/y;
    SWAP(x, y);
    x=x+y; y=x-y; x=x-y;
    printf ("X=%d, Y=%d", x, y);
    return 0;
}
```

- **A**
X=0,Y=0
- **B**
X=10,Y=20
- **C**
X=20,Y=10
- **D**
Compile time error

Correct Answer :A

Explanation

Let's go step by step:

1. $x = 10;$
2. $y = 20;$
3. $x=x*y ; x=200$
4. $y=x/y ; y=10$
5. $x=x/y; x=30$
6. Then control goes to, #define SWAP(a, b) {b ^= b; a ^= a; b ^= b ;} statement
7. Here, we are performing bitwise exclusive OR operation.
8. Results of above operation is: $a=0 , b=0$
9. $x=x+y; y=x-y; x=x-y;$ does not change anything because addition and subtraction does not affect variables if both the variables have 0 value.
10. Final result: $x=0 y=0$

#113 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
#define exp(a) a+a * 5 / a*a
int main( void )
{
    int x = exp(3+2) * 5;
    printf("Value of X=%d",x);
    return 0;
}
```

- A
Value of X=27

- **B**
Value of X=32
- **C**
Value of X=28
- **D**
Value of X=26

Correct Answer :A

Explanation

$$3+2+3+2*5/3+2*3+2*5 = 27$$

#114 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
#define int char
int main( void )
{
    int* i = 65, j=56;
    printf("sizeof(i)=%d sizeof(j)=%d", sizeof(i), sizeof(j));
    return 0;
}
```

- **A**
sizeof(i)=8 sizeof(j)=1
- **B**
sizeof(i)=4 sizeof(j)=1
- **C**
sizeof(i)=8 sizeof(j)=8
- **D**
sizeof(i)=4 sizeof(j)=4

Correct Answer :A

Explanation

Here the change lies within the declaration statement. int * i will declare a pointer of size 8 whereas j is a variable of char type and size as 1 byte.

#115 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
#define f1(para1,para2) para1##para2
#define f2(para2,para1) para1##para2
int main(void)
{
    char var_='A';
    printf("%c ",++f1(var,_));
    printf(" %d",--f2(_ ,var));
    return 0;
}
```

- **A**
B 65
- **B**
A 65
- **C**
C 65
- **D**
B 66

Correct Answer :A

Explanation

We can concat variables using preprocessor directive ## (for ex.. x##y = xy)

Here var_ is a variable name in which char ‘A’ is stored. In first printf() we are passing “var” and “_” as 2 parameters to f1 which will be concatenated by ## and becomes the original variable itself “var_”

Now var_ is storing A and we are pre incrementing it. So first printf will print B (character printed due to %c)

In second printf() due to pre decrementation value becomes A again but due to %d ASCII value of A i.e. 65 printed

#116 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#include<string.h>
int main(void)
{
    char joining1[10]="",joining2[10]:"",joining3[10]++;
    strcat(joining1,"TIME");
    strcat( joining1,"_"); strcat( joining1,"_");
    strcat( joining3,joining1);
    strcat( joining2,"_"); strcat( joining2,"_");
    strcpy( joining1,joining2);
    strcat( joining1,joining3);
    printf("\n time= %s", joining1);
    return 0;
}
```

- **A**
time= __TIME__
- **B**
time= current time will print

- **C**
compile time error
- **D**
run time error

Correct Answer :A

Explanation

strcat() - It is string concatenation function which will append the string at the end of existing string (if any)

strcpy() - It is a string copy function. The strcpy() function copies the string pointed by source (including the null character) to the destination.

1] joining1 = TIME

2] joining1 = TIME_

3] joining1 = TIME__

4] joining3 = TIME__

5] joining2 = _

6] joining2 = __

7] joining1 = __

8] 7] joining1 = __TIME__

ANS : time = __TIME__

#117 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
#include<stdlib.h>
int main ( void )
{
    char *title=NULL;
    title = (char *) malloc(15);
    strcpy(title, "C Programming");
    printf("String = %c", *title);
    free(title); title=0;
    strcpy(title, "C++");
    printf(" %s", title);
    return 0;
}
```

- **A**
String = C Programming
- **B**
Complile time error
- **C**
String = C C++
- **D**
exit value -1

Correct Answer :D

Explanation

Here “title” is a char pointer which was freed in between the program and then tried to access it again. This kind of pointer is called a “dangling pointer.”

Accessing a dangling pointer will result in undefined behavior.

It will not throw a compiler error but causes runtime error as segmentation fault and core dumped. When such error occurs program returns -1 to OS

#118 Explained Report Bookmark

What will be the output of the following C code? consider 32 bit system

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    char *ptr=NULL;
    ptr = (char *)calloc(20,sizeof(char));
    printf("%d bytes\n", sizeof(ptr));
    free(ptr); ptr=NULL;
    return 0;
}
```

- **A**
4 bytes
- **B**
1 bytes
- **C**
8 bytes
- **D**
20 bytes

Correct Answer :A

Explanation

The size of a pointer in C/C++ is not fixed. It depends upon different issues like Operating system, CPU architecture etc. Usually it depends upon the word size of underlying processor for example for a 32 bit computer the pointer size can be 4 bytes for a 64 bit computer the pointer size can be 8 bytes

#119 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    char *ptr=NULL;
    ptr = (char *)calloc(1,10);
    strcpy(ptr, "Sunbeam");
    ptr = (char *)realloc(ptr,20);
    strcat(ptr, " IT PARK");
    printf("%c",
        (ptr[0]>=65 && ptr[0]<=90) ? ptr[14]+32 : ptr[0]-32);
    free(ptr);
    ptr=NULL;
    return 0;
}
```

- A
N
- B
k
- C
o
- D
O

Correct Answer :B

Explanation

Explanation:-

calloc → allocates '1' block of size 10 bytes.

realloc → reallocates memory of 20 bytes to 'ptr'.

'ptr' = Sunbeam (after calloc)

'ptr' = Sunbeam IT PARK (after strcat)

1 2 3 4 5 6 7 8 9 10 11 12 13 14

ptr[0] → 'S' ascii → between 65 & 90 hence 'True'



ptr[14] + 32

ascii(K) + 32

75 + 32 = 107

↙
K

#120 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    char *ptr=NULL;
    strcpy(ptr , "Demo1");
    strcpy(ptr , "Demo2");
    free(ptr);
    printf("%s\n",ptr);
    return 0;
}
```

- A
Demo1

- **B**
Demo2
- **C**
Demo1Demo2
- **D**
exit value -1

Correct Answer :D

Explanation

Accessing a pointer which is free is illegal and gives error as it does not point to any location

#121 [Explained](#) [Report](#) [Bookmark](#)

what type of data u can store in this block of memory?

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    void *ptr=NULL;
    ptr = malloc(10);
    return 0;

}
```

- **A**
int
- **B**
char
- **C**
float

- **D**
all of above data types

Correct Answer :D

Explanation

The void pointer within C is a pointer that is not allied with any data types. This points to some data location within the storage means points to that address of variables. It is also known as a general-purpose pointer. In C, malloc() and calloc() functions return void * or generic pointers.

#122 [Explained](#) [Report](#) [Bookmark](#)

Which of the above three functions are likely to cause problems with pointers?

```
int * fun1 (void)
{
    int x= 10;
    return (&x);
}
int * fun2 (void)
{
    int * px;
    *px= 10;
    return px;
}
int *fun3 (void)
{
    int *px;
    px = (int *) malloc (sizeof(int));
    *px= 10;
    return px;
}
```

- A
function fun1 and fun2
- B
function fun2 and fun3
- C
function fun1 , fun2 and fun3
- D
function fun3

Correct Answer :A

Explanation

Function pointers can be useful when we want to create callback mechanism, and need to pass address of a function to another function. They can also be useful when we want to store an array of functions, to call dynamically.

fun1 : Here the function is returning address of the variable x (& x) but the return type is pointer to integer not address. So incorrect.

fun2 : *px = 0 directly assigned a value but still px doesn't point to any memory location, so memory initialization or allocation should be done before. So incorrect.

fun3 : Correction made in fun2, memory pre allocated, So correct

#123 **Not Explained Report Bookmark**

What will be the output of the following C code?

```
#include <stdio.h>
void print(double a[])
{
    int n=sizeof(a)/sizeof(*a)+sizeof(&a)-(a[7] /1.1f);
```

```

int i;
for (i = 0; i < n; i++)
printf(" %.1lf ", a[i]);
return;
}
int main( void )
{
    double arr[] = {1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8} ;
    print(arr);
    return 0;
}

//note :: consider 64 bit compilation.

```

- **A**
1.1
- **B**
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 0.0
- **C**
1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 garbage
- **D**
Compile time error

Correct Answer :A

No Explanation Available

#124 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include <stdio.h>
#define SIZE(arr) sizeof(arr) / sizeof(*arr);
void fun(int* arr, int n)
{
    *arr += *(arr + n - 1) += *arr;
}
void printArr(int* arr, int n)

```

```

{
    int i;
    for(i = 0; i < n; ++i)
        printf("%d ", arr[i]);
    return;
}

int main()
{
    int arr[] = {1, 2, 3};
    int size = SIZE(arr);
    fun(arr, size);
    printArr(arr, size);
    return 0;
}

```

- **A**
2 4 5
- **B**
5 2 4
- **C**
2 4 10
- **D**
compile time error

Correct Answer :B

Explanation

The crux of the question lies in the expression: `*arr += *(arr + n - 1) += *arr;`

The composite operator (here `+=`) has right to left associativity.

First 1 is added to the last element of the array.

The result is then added to the first element of the array.

#125 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main( void )
{
    int arr[3]={10,20,30,40};
    --2[arr];
    printf("%d", - --2[arr]);
    return 0;

}
```

- **A**
-29
- **B**
-28
- **C**
Compiler error
- **D**
28

Correct Answer :B

Explanation

--2[arr];

Pre-decrement operator is used here i.e. first decrement operation will be performed and then value will be passed for next step.

--2[arr] means decrement 2nd element of array arr by 1 and store it in 2nd location only i.e. original location as it was before. Here 2nd element is 30, which becomes 29 after performing pre-decrement operation

We are performing above step again during print operation which makes 2nd element 28 from 29.

```
printf("%d", -2[arr]);
```

It will print the output of step 3 along with '-' sign

#126 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    int arr[5]={5-3*0-1,10,15,20,25};
    printf("%8d",arr[arr[0]]);
    printf("%8d",arr[arr[1-1]]);
    printf("%8d",arr[arr[1*0]]);
    printf("%8d",arr[arr[0/1]]);
    return 0;
}
```

- **A**
25 25 25 25
- **B**
25 4 4 4
- **C**
Compiler error
- **D**
4 4 4 4

Correct Answer :A

Explanation

25 25 25 25

1. arr[0] i.e. $5-3*0-1$ results in 4 thus arr[0] holds 4.
2. arr[arr[0]] is nothing but arr[4] which is 25
3. arr[arr[1-1]] is nothing but arr[4] which is 25
4. arr[arr[1*0]] is nothing but arr[4] which is 25
5. arr[arr[0/1]] is nothing but arr[4] which is 25

#127 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    int a[] = {45, 23, 43, 21, 98, 67};
    int *ptr = (int*)(&a+1);
    printf("%d ", *(ptr-sizeof('0')-1));
    return 0;
}
```

- **A**
23
- **B**
21
- **C**
67
- **D**
compile time error

Correct Answer :A

Explanation

```
int *ptr = (int*)(&a+1);
```

Above statement will point to the next location of the last element held by a array.

```
printf("%d ", *(ptr-offsetof('0')-1));
```

`sizeof('0')` is 4Byte. Thus `*ptr` location which is 6 will subtract 4 from it resulting in 2. `*(ptr-offsetof('0'))` is location 2nd i.e. 43. `*(ptr-offsetof('0')-1)` will point to the location 1st i.e. 23

Thus 23 will be printed on screen.

#128 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main( void )
{
    int a[100]={-1},i=a[1];
    if(!(*(&a[0]+i)==i[0+i+a]))
    {
        printf("Welcome to ccat@ online");
    }
    else
    {
        printf("Welcome to ccat @ offline");
    }
    return 0;
}
```

- **A**
Welcome to ccat @ offline
- **B**
Welcome to ccat @ online
- **C**
compile time error
- **D**
run time error

Correct Answer :A

Explanation

Here is a partially initialized array. The remaining elements of partially initialized array are set to 0. According to this rule $a[1] = 0$. $i=0$.

$(a+0+i) = a[0]$ as $i=0$

$i [0+i+a] = a[0]$

The value at $a[0]$ is compared with $a[0]$. Hence the condition is true. But $!$ is applied. So true will become false. And hence the statement in the else condition will be printed.

#129[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main( void )
{
    int a[10]={1,2,3,4,1}, i,j,m;
    i=a[1]++ + ++a[2] + ++a[3];
    j=a[2]++ + --a[3] - a[4]--;
    m=++a[j];
```

```

    printf ("%d,%d,%d", i, j, m);
    return 0;
}

```

- **A**
3,7,11
- **B**
1,7,11
- **C**
11,7,3
- **D**
11,7,1

Correct Answer :D

Explanation

1. $i=a[1]++ + ++a[2] + ++a[3];$

$a[1]++$ will pass 2, $++a[2]$ will increment 3 by 1 and pass 4, $++a[3]$ will increment 4 by 1 and pass 5. After adding 2+4+5 we get 11

1. After the first pass the array will be [1,3,4,5,1]
2. $j=a[2]++ + --a[3] - a[4]--;$

$a[2]$ pass 4, $--a[3]$ pass 4 by decreeing 5 to 4, $a[4]--$ will pass 4. After performing operation 4+4-1 we get 7

1. $m=++a[j];$

here $a[j]$ means $a[7]$ which does not hold any value or we can say it is 0. But here we are performing pre increment, thus $++a[j]$ will increment 0 to 1 and store it in m.

#130 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main( void )
{
    double arr[]={1.2,2.3,3.4,4.5,5.6,6.7,7.8};
    float size=(float)(sizeof(arr)+1)/(sizeof(*arr));
    printf("%.4f",size);
    return 0;
}

//note :: consider 64 bit compilation.
```

- **A**
57.0000
- **B**
7.0000
- **C**
7.1250
- **D**
8.0000

Correct Answer :C

Explanation

The size of double is 8 Byte. There are 7 elements in array arr[], so total memory consumed is $7 \times 8 = 56$ Bytes. $\text{sizeof}(arr) + 1$

here we are adding 1 in the size calculation statement which makes it 57. $\text{sizeof}(*\text{arr})$ here size of a single or lets say first double element is 8 Byte.

we then perform operation of division (57/8) which results in 7.1250 and that too there are 4 digits after '.' because we have mentioned it in
printf("%.4f",size);

#131 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>
#include<string.h>
int main(void)
{
    char* convert(char *s);
    char str1[] = "strings";
    char str2[] = "STRINGS";
    if (strcmp(convert(str1), convert(str2))==0)
        printf("%s Strings are equal %s", str1, str2);
    else
        printf("%s Strings are not equal %s", str1, str2);
    return 0;
}
```

- **A** STRINGS Strings are not equal strings
- **B** strings Strings are not equal STRINGS
- **C** Compile time error
- **D** STRINGS Strings are equal strings

Correct Answer :C

Explanation

the reference to convert(char*) is undefined and thus it returns exit status 1 instead of 0.

#132 Explained Report Bookmark

The correct statement to copy string literal constant

“Hello” to string str is?

- **A**
str=“Hello”
- **B**
strcpy(“Hello”,str);
- **C**
strcpy(str,”Hello”)
- **D**
none

Correct Answer :C

Explanation

The function prototype of strcpy() is:

```
char* strcpy(char* destination, const char* source);
```

- The strcpy() function copies the string pointed by source (including the null character) to the destination.

The strcpy() function also returns the copied string

#133 Explained Report Bookmark

What will be the output of the following C code?

```

#include<stdio.h>
int main(void)
{
    char s[] = "Sunbeam", ch;
    int i=0;
    ch = s[i++];
    printf("%c", ch);
    ch = s[++i];
    printf("%c", ch);
    ch = ++i[s];
    printf("%c", ch);
    ch = i++[s];
    printf("%c", ch);
    return 0;
}

```

- **A**
Snoo
- **B**
Soon
- **C**
snoo
- **D**
Sono

Correct Answer :A

Explanation

Basics of the program is, `++i` is a pre increment operator which first increments and then passes the value whereas `i++` is a post increment operator which first passes the value and then increments it.

Original value of `i=0`

S[i++]: access S[0] then increment i to 1. (S)

S[++i]: increment i to 2 then access S[2] (n)

++i[S]: since [] has higher precedence than ++. The program will work on ASCII values. Here the value located is 'n', which will become 'o' after performing the increment operation on n's ASCII value (o)

.i++[S]: this will work as above since there is a post increment operator used.(o)

#134 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    char dest[] = "Visual basic", src[] = "C++";
    puts(strcpy(&dest[7], src)-7);
    return 0;
}
```

- **A**
Visual C++
- **B**
C++
- **C**
basic
- **D**
Visual

Correct Answer :A

Explanation

strcpy returns starting address of new string

#135 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    char s []="Sunbeam Pune";
    printf("%c ",*(s+2));
    printf("%s ",s+5);
    printf("%s ",s);
    printf("%c ",*(s+2));
    return 0;
}
```

- **A**
Compile time error
- **B**
Run time error
- **C**
Prints array elements
- **D**
None of the above

Correct Answer :C

Explanation

Basics of this program is %c prints a character and %s prints a string which is a collection of character and whitespace in this case.any kind of array starts with 0th position thus index of last element will be n-1th.

For example,

1 2 3 4 5 : here index of 1 is 0th and index of 5 is 4

```
printf("%c ",*(s+2));
```

:character at 2nd index

```
printf("%s ",s+5);
```

:substring of s array starting including and from 5th index to last index

```
printf("%s ",s);
```

:print char array s as a string

```
printf("%c ",*(s+2));
```

:character at 2nd index

#136Not Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
#define COL 3
#define ROW 2
int main( void )
{
    int arr[ROW][COL] = {10,20,30,40};
    int *ptr[] = {(int *)arr+2, (int *)arr+1, (int *)arr};
    printf("%d %d %d %d\n", ptr[0][1], *(*(ptr + 1) + 0),
           *(ptr + 0)[2], *(ptr[1] + 1));

    return 0;
```

}

- A
40 20 10 0
- B
40 20 30 10
- C
40 20 10 30
- D
40 20 30 0

Correct Answer :C

No Explanation Available

#137 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of following program if base address of arr is 4289999264.

```
#include<stdio.h>
int main(void)
{
    int a[2][2] = {{1,2},{3,4}};
    printf("%u %u %u %u\n", a+1, &a+1, (a+1), &(a+1));
    return 0;
}
```

- A
4289999272 4289999280 4289999272 4289999280
- B
4289999268 4289999280 4289999268 4289999280
- C
4289999272 4289999272 4289999272 4289999272
- D
Compile time error

Correct Answer :D

Explanation

None of the above

#138 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    int a[2][2] = {{1,2},{1,2}}, r,c;
    for(r=0; r<2; r++)
        for(c=0; c<2; c++)
            printf("%d %d %d %d\n",r,c,*(*(a+r)+c), *(*(a+c)+r));
    return 0;
}
```

- **A**
0 0 1 1
0 1 2 1
1 0 1 2
1 1 2 2
- **B**
0 0 1 1
0 1 1 2
1 0 2 1
1 1 2 2
- **C**
0 0 1 1
0 1 2 2
1 0 1 1
1 1 2 2

- D
0 0 1 1
0 1 1 1
1 0 2 2
1 1 2 2

Correct Answer :A

Explanation

locations:

a[0][0]:1 :6422016

a[0][1]:2 :6422020

a[1][0]:1 :6422024

a[1][1]:2 :6422028

$*(a+r)+c : 6422016$

$*(a+c)+r : 6422016$

Thus according to location: 0 0 1 1

$*(a+r)+c : 6422020$

$*(a+c)+r : 6422024$

Thus according to location: 0 1 2 1

`*(a+r)+c : 6422024`

`*(a+c)+r : 6422020`

Thus according to location:1 0 1 2

`*(a+r)+c : 6422028`

`*(a+c)+r : 6422028`

Thus according to accessed location:1 1 2 2

As you can see here. How the locations are called. Here make that even though we have + operator in the print method we are not exactly printing the incremented number here, but we are actually accessing the values stored in ** location.

#139 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};
    int *ptr_a = &a[1][0];
    int **ptr_ptr = &ptr_a;
    printf("%d %d %d\n", **ptr_ptr, *ptr_a, **a);
    return 0;
}
```

- A
111
- B
444

- **C**
441
- **D**
141

Correct Answer :C

Explanation

Here, `*ptr_a = &a[1][0]` ;location `a[1][0]` which is 4 is pointed by `ptr_a` pointer.

`**ptr_ptr = &ptr_a` ; `ptr_a` pointer address is pointed by double pointer `ptr_ptr` i.e. value of both `**ptr_ptr` and `*ptr_a` is same (4).

Then a pointer is pointing to the first element of an array which is 1.

Thus printed 4 4 1

#140 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    int arr[2][3] = {1,2,3,4,5},row,col;
    for(row=0; row<3; row++)
        for(col=0; col<2; col++)
            printf("%d",arr[row][col]);
    return 0;
}
```

- **A**
012345

- **B**
123450
- **C**
12345
- **D**
1245 [garbage value] [garbage value]

Correct Answer :D

Explanation

0	1
2	
1	3
2	
4	0

row = 0

col = 0 --- arr [0][0] = 1

col = 1 --- arr [0][1] = 2

row = 1

col =0 --- arr [1][0] = 4

col =1 --- arr [1][1] = 5

row = 2

col =0 --- arr [2][0] = garbage value

col =1 --- arr [2][1] = garbage value

Note : row = 2 not available in arr[2][3] so it will print garbage value. Some compilers can show ans 00 instead garbage as 0 can be one of the garbage value but the ans varies from compiler to compiler.

#141 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    char arr[4][8] = {"PG-DAC", "PG-DESD", "PG-DBDA"};
    printf("%c%s", **arr, *(arr+1)+1);
    return 0;
```

}

- **A**
PPG-DAC
- **B**
PPG-DESD
- **C**
PPG-DBDA
- **D**
PG-DESD

Correct Answer :D

Explanation

`**arr` will point to the base address of array i.e. `arr[0][0]`

The content at `arr[0][0] = P`

So `%c` in `printf` will print `P`

`*(arr +1) +1 = *(arr +1)` means `arr[1]`

The string present at `arr [1] = “ PG-DESD ”`

We are adding 1 to `arr[1]` so the characters from `arr[1][1]` will get printed until they encounter a NULL character.

So `%s` in `printf` will print `G-DESD`

Final ans becomes PG-DESD

#142 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    char arr[5][8] = {"DAC", "DESD", "DMC", "DBDA", "PreCAT"};
    char *ptr = arr[4];
    printf("%c.%s\n", *(ptr+3) + *(ptr+3) - ptr[4],
           (ptr+3) - *(ptr+1) + ptr[1]);
    return 0;

}
```

- **A**
P.CAT
- **B**
E.CAT
- **C**
C.CAT
- **D**
None of the above

Correct Answer :B

Explanation

$*(\text{ptr}+3)$ means ASCII value of the third character of PreCAT.

Ptr[4] means ASCII value of the fourth character of PreCAT.

Then the first printed character will be, $67+67-65=69$. Character of ASCII value 69 is E.

Value of *(ptr+1) and ptr[1] is 114 which gets cancelled. Then (ptr+3) is a string i.e. PreCAT where string from location 3rd i.e. CAT.

Thus printed E.CAT

#143 **Explained Report Bookmark**

What will be the output of the following C code?

```
#include<stdio.h>
#define so sizeof
int main(void)
{
    char s[4][32];
    printf("%d %d %d", so(s[2][2]), so(s[2]), so(s));
    return 0;
}
```

- **A**
1 4 128
- **B**
1 4 64
- **C**
1 32 128
- **D**
1 32 64

Correct Answer :C

Explanation

so macro defined for sizeof

Size of entire array = No. of rows * No. of columns * Size of each element

sizeof (s) = 4 * 32 * 1 = 128

Size of 1 row = No. of columns * Size of each element

Sizeof (s[2]) = 32 * 1 = 32

Size of 1 element = 1 (because it is character array)

#144 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
struct s1
{
    char a[4];
    char *p;
}
o = {"DAC", "DMC"};
int main(void)
{
    o.p = o.a+1;
    printf("%c %s\n", *o.p, o.a);
    return 0;
}
```

- **A**
M DAC
- **B**
A DAC
- **C**
A DMC

- D
M DMC

Correct Answer :B

Explanation

struct s1 will create a structure where char array of 4 size and char pointer is defined.

o = {"DAC","DMC"}; will assign the values to first and second location of array a where o is a list.

o.p = o.a+1; here o.a is “DAC” and +1 means character at 1st index which is ‘A’. It will be stored at the location pointed by o.p.

*o.p,o.a. Here p is a pointer thus o.p will print the value pointed by o.p and o.a is the first element of the array a..

#145 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
struct s2 {
    char *cp;
    struct s1{
        char a[4];
        char *p;
    }o1;
}o2;
int main(void)
{
    printf("%d %d %d\n",sizeof(struct s2) , sizeof(o2) ,
```

```
    sizeof(o2.o1));
    return 0;
}
```

- A 12 12 8
- B 24 24 16
- C Both A and B
- D None of the above

Correct Answer :B

Explanation

the size of o1 i.e. struct s1 is 16 because of char a[4]; char *p;. Struct itself hold 3 Byte and remaining will be held by mentioned members.

Struct s2 and o2 are similar as you can see in the program. Struct s2 has struct s1 held inside it.

Thus s2 result will be equal to 16 of o1 and char *cp; resulting 24.

#146 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
struct s2 {
    char *cp;
    struct s1{
```

```

        char a[4];
        char *p;
    }o1;
}
o2 = {"DAC", "DESD", "DMC"};
int main(void)
{
    printf("%s %s\n", ++o2.cp, ++o2.o1.p);
    return 0;
}

```

- **A**
AC MC
- **B**
AC ESD
- **C**
MC AC
- **D**
ESD MC

Correct Answer :A

Explanation

`o2.cp` will access first element of the array i.e. DAC, but since we are pre incrementing the value , only those will be printed which are no at 0th index which is AC. `*p` is the pointer responsible for handling the last element of `o1` union that of `o2` array. Considering how AC was printed on screen we can say that after performing pre incrementing MC of DMC will be printed on screen.

#147 Not Explained Report Bookmark

What will be the output of the following C code?

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

```

int main(void)
{
    struct s{
        char *p;int i;
        struct s *sp;
    }
    a[] = {"abcd",1,a+1,"efgh",2,a+2,"ijkl",3,a}, *p;
    p = a;
    printf("%s %s %s\n",a[0].p,p->p,a[2].sp->p);
    return 0;
}

```

- **A**
abcd abcd ijkl
- **B**
abcd efgh ijkl
- **C**
abcd abcd efgh
- **D**
abcd abcd abcd

Correct Answer :D

No Explanation Available

#148 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include<stdio.h>
struct s {
    int i;
    struct s obj;
}s1;
int main(void)
{
    s1.i = 100;
}

```

```

s1.obj = s1;
printf("%d", s1.obj.i);
return 0;

}

```

- **A**
100
- **B**
Garbage value
- **C**
Compiler error
- **D**
Run time error

Correct Answer :C

Explanation

The compile time error will be thrown i.e. field 'obj' has incomplete type. The error means that you try to add a member to the struct of a type that isn't fully defined yet, so the compiler cannot know its size in order to determine the object's layout. In your particular case, you try and have the struct Cat hold a complete object of itself as a member (the mother field).

#149 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include<stdio.h>
union
{
    short i;
    char c;
}u;
int main(void)
{

```

```

    u.c = 'D';
    u.i = 0x0041;
    printf("%d %c", sizeof(u), u.c);
    return 0;

}

```

- **A**
2A
- **B**
2D
- **C**
3A
- **D**
3D

Correct Answer :A

Explanation

u is holding a character which is a size od 2 byte, thus 2 will be printed on screen. Then since we are union i.e. (It is a collection of variables of different datatypes in the same memory location. We can define a union with many members, but at a given point of time only one member can contain a value.) D will be converted in different form resulting ascii value from hexadecimal number i.e A.

#150 **Not Explained Report Bookmark**

What will be the output of the following C code?

```

#include<stdio.h>
union u{
    int i;
    char c[4];
};
int main(void)

```

```

{
union u u1;
u1.i=0;
u1.c[1] = u1.c[2] = 'F';
printf("%s",u1.c);
return 0;

}

```

- **A**
no output return value from main function is zero
- **B**
Garbage character followed by 'FF'
- **C**
FF
- **D**
Compile Error

Correct Answer :A

No Explanation Available

#151 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include<stdio.h>
#pragma pack(1)
struct
{
    char ca[10];
    union u
    {
        int i;
        char c;
        long int l;
    }
}
```

```
    }u1;
}s1;
int main(void)
{
    printf("%d", sizeof(s1) + sizeof(s1.u1));
    return 0;
}

//consider 32 bit compilation.
```

- A 20
- B 26
- C 23
- D 18

Correct Answer :B

Explanation

size of s1 is 18 and the size of union u1 within structure s1 is 8.

Here total size consumed by variables mentioned inside the u1 union will hold 8 bytes of memory. S1 struct holds u1 union

i.e 8byte and char array of 10 elements where each element holds 1 byte which makes the total 18 bytes of size.

its a compiler-based question some will give an answer as 18 on 32 bit compiler

#152 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    enum colours {RED,BLACK,WHITE=5,YELLOW,BLUE,GREY};
    printf("%d %d %d %d",RED,YELLOW,WHITE,GREY);
    return 0;
}
```

- A
1 2 3 4
- B
0 2 3 4
- C
1 6 7 8
- D
0 6 7 8

Correct Answer :D

Explanation

```
enum colours {RED,BLACK,WHITE=5,YELLOW,BLUE,GREY};
```

This statement is holding total 6 elements or values. But since white is allocated the value 5, automatically value of rest of the non preceded elements gets increased that is if white=5 then yellow=6, blue=7, grey=8 and so.

#153 Explained Report Bookmark

What will be the output of the following C code?

```

#include<stdio.h>
int main(void)
{
    enum choice {CH1, CH2, CH3};
    enum choice ch1, ch2, ch3;
    ch1 = CH1;
    ch2 = CH3;
    ch3 = CH2;
    printf("%d, %d, %d,", ch1, ch2, ch3);
    return 0;
}

```

- **A**
0, 1, 2
- **B**
1, 2, 3
- **C**
0, 2, 1
- **D**
1, 3, 2

Correct Answer :C

Explanation

We have declared an enum with CH1 , CH2 and CH3 which holds it's index as value i.e. 0,1,2 respectively.

1. ch1 = CH1;

Assigning CH1=0 to ch1

1. ch2 = CH3;

Assigning CH3=2 to ch

1. ch3 = CH2;

Assigning CH2=1 to ch3

1. Thus resulting final values held by ch1=0, ch2=2 and ch3=1

#154 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    enum choice {CH1, CH2, CH3};
    enum choice ch1, ch2, ch3;
    ch1 = CH1;
    ch2 = CH3;
    ch3 = ch2-ch1;
    printf("%d %d", sizeof(enum choice), ch3);
    return 0;
}
```

- **A**
4 2
- **B**
2 2
- **C**
4 1
- **D**
2 1

Correct Answer :A

Explanation

Here enum is treated as integer.

Thus CH1, CH2 and CH3 values are 0, 1 and 2 respectively since their index value is it.

1. ch1 = CH1;

Assigning CH1 i.e. 0 to ch1

1. ch2 = CH3;

Assigning CH2 i.e. 1 to ch2

1. ch3 = ch2-ch1;

Assigning CH3 i.e. to ch2-ch1 which is 2-0=2

1. sizeof(enum choice) depends on compiler only. Even if enum has 10 elements in it, common compiler would have displayed 4 as output. Thus in most of the cases 4 will be output.

#155 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    typedef int int_t;
    int_t *iptr;
    int ival = 60;
    iptr = &ival;
```

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

- **A**
60
- **B**
Compiler error
- **C**
Linker error
- **D**
Runtime error

Correct Answer :A

Explanation

t is very simple program of using typedef and pointer.

1. `typedef int int_t;`

whenever we use `int_t` during our program it will act as `int` such as if we type “`int_t e=3;`” compiler will create a variable `e` of integer type which is holding 3 value.

1. `int_t *iptr;`

Declaring pointer

1. `int ival = 60;`

Declaring variable `ival`

1. iptr = &ival;

Assigning pointer to a variable

1. printf("%d", *iptr);

Printing value stored at “ival” using pointer “*ptr”

#156 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
struct time {
    int ss:7;
    int mm:7;
    int hh:4;
};
int main(void)
{
    struct time t1;
    printf("%d %d", sizeof(t1), sizeof(t1.ss));
    return 0;
}
```

- A
4
- B
Compiler error
- C
Runtime error
- D
no output

Correct Answer :B

Explanation

a bit field is a data structure that allows the programmer to allocate memory to structures and unions in bits in order to utilize computer memory in an efficient manner. Thus, We can not calculate the size of the bit field in c using the sizeof operator.

#157 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    unsigned short int a = 10;
    a = ~a;
    printf("%u\n", a);
    return 0;
}
```

- **A**
-11
- **B**
65525
- **C**
65526
- **D**
-9

Correct Answer :B

Explanation

`~` operator is used for bitwise complement operations.

1. Value of `a` is 10
2. After conversion using bitwise complement it is replaced by -11 which is a signed integer
3. In print statement `%u` i.e. unsigned integer is expected to be printed. Thus signed -11 is converted to unsigned int which is 65525

#158 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    int a = 144;
    if (a = a >> 4)
        printf("a=%d\n", a=a<<3);
    return 0;
}
```

- A
72
- B
144
- C
288
- D
36

Correct Answer :A

Explanation

`>>` and `<<` is a bitwise shift operator which works on binary digits.

1. $a = a >> 4$ Here binary right shift operation is performed on 144 that is 10010000 is shifted right by 4 digits which makes it 00001001 i.e. 9 in decimal
2. $a=a<<3$ Here binary left shift operation is performed on 9 i.e. 00001001 by shifting it's digits to left by 3 positions. It makes the a value hold 1000111 i.e. 72
3. Note that whenever left shift is performed 1 is added from the right end and for right shift 0 is added from the left end.

#159 Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>
int main(void)
{
    if (!(7 | 8))
        printf("Honesty");
    if ((~7 | 0x000f) != 8)
        printf("is the best policy\n");
    return 0;
}
```

- **A**
Honesty is the best policy
- **B**
Honesty
- **C**
is the best policy
- **D**
run time error

Correct Answer :C

Explanation

1. if (!(7 | 8))

Bitwise or operation is performed. Since 7 bitwise or 8 doesn't result true, the if block will not be performed

1. printf("Honesty");

This block will not be executed since if block return false

1. if ((~7 | 0x000f) != 8)

Here bitwise operation is performed between ~7 i.e. -8 and 0x000f hexadecimal i.e. 15 in decimal which results in -1.

1. printf("is the best policy\n");

Since if block return true, this print statement gets executed

#160 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    char var=0x04;
    var = var | 0x04;
    printf("%d,",var);
    var |= 0x01;
    printf("%d",var);
    return 0;
}
```

- A
8,9

- **B**
4,5
- **C**
8,8
- **D**
4,4

Correct Answer :B

Explanation

Value of var is 0x04 (0100), Consider the expression var = var | 0x04 The OR (|) of 0100, 0100 is 0100, hence value will remain 0100. After the expression var |=0x01, value will be 0101 that is 0x05.

#161 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    int x=10; x &= ~2;printf("x = %d",x);
    return 0;
}
```

- **A**
x = 10
- **B**
x = 8
- **C**
x = 12

- **D**
 $x = 0$

Correct Answer :B

Explanation

1. ~ 2 results in -3 value.
2. $x \&= (-2)$ perform bitwise AND operation where The result of AND is 1 only if both bits are 1.
3. Then by performing the operation $x \& (-2)$ we get 8 where x was 10.

#162 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    char flag=0x0f;
    flag &= ~0x02; printf("flag = %d",flag);
    return 0;
}
```

- **A**
flag = 13
- **B**
flag = d
- **C**
flag = 22
- **D**
flag = 10

Correct Answer :A

Explanation

0x0f is blank in char and 15 in decimal ASCII value. ~0x02 is 2 in decimal and blank in char ASCII value on which we are performing “~” bitwise operation resulting in -3 on which we are performing bitwise AND operation.resulting in 13 i.e. 1101 & 1111 = 1101

#163 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    FILE *fp = stdout;
    int num=102;
    fprintf(fp, "%d-%c-%d-%c", num, num-32, num+=32, num-=32);
    return 0;
}
```

- **A**
102-F-102-f
- **B**
102-f-70-F
- **C**
70-f-102-F
- **D**
70-F-70-f

Correct Answer :A

Explanation

```
1. FILE *fp = stdout;
```

*fp is a file pointer which will hold the reference to the opened(or created) file. Here we are using it to use standard output as file.

```
1. int num=102;
```

Declaring num to 102.

```
1. fprintf(fp, "%d-%c-%d-%c",num,num-32,num+=32,num-=32);
```

fprintf allows you to "write" information to the screen for the user to view.

Here we are printing 2 integers and 2 characters.

num: holds 102

num-32: the output will be (num-32)=(102-32)=’F’ since the 70 is the ASCII value for it. num+=32: interpret as num=num+32, here only 102 i.e. original input will be printed

num-=32: interpret as num=num-32, here’s a twist. The ASCII value of num i.e. 102 will be printed here which is ‘f’.

#164[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main(void)
{
    printf("\?!\n");
    return 0;
```

}

- **A**
Compile Time Error
- **B**
Run Time Error
- **C**
??!
- **D**
\?\?!\n

Correct Answer :C

Explanation

\? is one of the escape characters. It is used for the presentation of trigraph.

#165Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = -3;
    int k = i % 2;
    printf("%d\n", k);
```

- **A**
Compile time error
- **B**
-1
- **C**
1

- **D**
none

Correct Answer :B

Explanation

In this program k will get -1 ,

$k = i \% 2$, $i = -3$ so $k = -3 \% 2$ (according to the precedence) it evaluate like this

$-(3 \% 2) = -(1.5)$ but because of integer the output is -1

#166 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 3;
    int l = i / -2;
    int k = i % -2;
    printf("%d %d\n", l, k);
    return 0;
}
```

- **A**
Compile time error
- **B**
-1 1

- **C**
1 -1
- **D**
none

Correct Answer :B

Explanation

In c language, by default integer is signed int which means the value of the variable can be positive, negative and zero

Here, we initialize i with 3, and execute the next line expressions

i= i / -2 --> 3/-2 will give -1.5 but because of int variable it takes only -1, and execute the next line

k= i % -2 --> 3 % -2 will give 1 as remainder and it assign in k variable and we get -1 1 as output from print statement

#167Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 5;
    i = i / 3;
    printf("%d\n", i);
    return 0;
}
```

- **A**

- **B**
1
- **C**
3
- **D**
error

Correct Answer :B

Explanation

```
#include <stdio.h>
int main()
{
    int i = 5;
    i = i / 3;
    //here i = 5/3 thats will return 1.666
    //but i has data type as integer so it will holds
    //only 1 not fractional part
    printf("%d\n", i);
    //final output will be 1
    return 0;
}
```

#168[Explained](#) [Report](#) [Bookmark](#)

What will be the final value of x in the following C code?

```
#include <stdio.h>
void main()
{
    int x = 5 * 9 / 3 + 9;
}
```

- **A**
3.75

- **B**
Depends on compiler
- **C**
24
- **D**
3

Correct Answer :C

Explanation

```
#include <stdio.h>
void main()
{
    int x = 5 * 9 / 3 + 9;
    /* has highest precedence so
     * 5 * 9 = 45 will evaluate first
     * then / has highest precedence so
     * 45/3 =15 after that 15 + 9 = 24
    printf("%d",x);
}
```

#169 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>
void main()
{
    int a = 3;
    int b = ++a + a++ + --a;
    printf("Value of b is %d", b);

}
```

- **A**
Value of x is 12

- **B**
Value of x is 13
- **C**
Value of x is 10
- **D**
Compilers Depend

Correct Answer :D

Explanation

The solution if this question will be different on different Compilers of C.

i.e. It is compiler dependent. But, in TurboC, it will return the value of b as 8.

Explanation: Evaluation will start from right to left

#170 [Explained](#) [Report](#) [Bookmark](#)

What is the precedence of arithmetic operators (from highest to lowest)?

- **A**
%, *, /, +, -
- **B**
%, +, /, *, -
- **C**
+, -, %, *, /
- **D**
%, +, -, *, /

Correct Answer :A

Explanation

All arithmetic operators in C language follow the left to right associativity.
Their precedence from highest to lowest is as given below:

() => Brackets

% => Modulus

* => Multiplication

/ => Division

+ => Addition

- => Subtraction

#171 [Explained](#) [Report](#) [Bookmark](#)

Which of the following is not an arithmetic operation?

- **A**
 $a^* = 10;$
- **B**
 $a / = 10;$
- **C**
 $a != 10;$
- **D**
 $a \% = 10;$

Correct Answer :C

Explanation

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

OPERATOR	MEANING OF OPERATOR
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

OPERATOR	EXAMPLE	SAME AS
-----------------	----------------	----------------

=	$a = b$	$a = b$
$+=$	$a += b$	$a = a+b$
$-=$	$a -= b$	$a = a-b$
$*=$	$a *= b$	$a = a*b$
$/=$	$a /= b$	$a = a/b$
$\%=$	$a \%= b$	$a = a \% b$

#172 [Explained](#) [Report](#) [Bookmark](#)

which of the following data type will throw an error on modulus operation(%)?

- **A**
char
- **B**
short
- **C**
int
- **D**
float

Correct Answer :D

Explanation

There is no technical reason why it wouldn't be possible to define the `%` operator for floating-point types. Here's what the C99 rationale says:

6.5.5 Multiplicative operators

The C89 Committee rejected extending the `%` operator to work on floating types as such usage would duplicate the facility provided by `fmod` (see §7.12.10.1).

And as mafso found later:

7.12.10.1 The `fmod` functions

The C89 Committee considered a proposal to use the remainder operator `%` for this function; but it was rejected because the operators in general correspond to hardware facilities, and `fmod` is not supported in hardware on most machines.

They seem somewhat contradictory. The `%` operator was not extended because `fmod` already filled that need, but `fmod` was picked to fill that need because the committee did not want to extend the `%` operator? They cannot very well both be true at the same time.

#173 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int main()

{

    int a = 10;

    double b = 5.6;

    int c;

    c = a + b;

    printf("%d", c);

}
```

- **A**
15
- **B**
16
- **C**
15 6
- **D**
10

Correct Answer :A

Explanation

```
#include <stdio.h>

int main()

{
```

```
int a = 10;

double b = 5.6;

int c;

c = a + b;

//here c data type is integer when we

// add a + b = 10 + 5.6 thats roundoff to 15

//so final output will be 15

printf("%d", c);

}
```

#174Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{

    int a = 10, b = 5, c = 5;

    int d;

    d = a == (b + c);
```

```
    printf ("%d", d);  
}
```

- **A**
Syntax error
- **B**
1
- **C**
10
- **D**
5

Correct Answer :B

Explanation

```
#include <stdio.h>  
  
int main()  
  
{  
  
    int a = 10, b = 5, c = 5;  
  
    int d;  
  
    d = a == (b + c);  
  
    // here brackets has highest associativity so  
  
    // b + c will evaluate first = 5 + 5 =10  
  
    //then result will compare to a thats means
```

```
//10 = 10 it will return true (1) stores in d  
  
printf("%d", d);  
  
//So final output will be 1  
  
}
```

#175 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>  
  
void main()  
  
{  
  
    int x = 1, y = 0, z = 5;  
  
    int a = x && y && z++;  
  
    printf("%d", z);  
  
}
```

- A
6
- B
5
- C
0
- D
none

Correct Answer :B

Explanation

For starters function main without parameters shall be declared like

```
int main( void )
```

In the program the initialization expression can be represented the same way as

```
int a = ( x && y ) && ( z++ );
```

According to the C Standard (6.5.13 Logical AND operator)

4. ...If the first operand compares equal to 0, the second operand is not evaluated.

As before the first operand (`x && y`) of the expression is equal tp 0 and according to the quote the second operand (`z++`) is not evaluated.

As result z will be equal to 5 as before.

#176 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
void main()
```

```
{  
  
    int x = 1, y = 0, z = 5;  
  
    int a = x && y && z++;  
  
    printf("%d", z);  
  
}
```

- **A**
6
- **B**
5
- **C**
0
- **D**
none

Correct Answer :B

Explanation

In the program

```
int a = ( x && y ) && ( z++ );
```

According to the C Standard (6.5.13 Logical AND operator)

If the first operand compares equal to 0, the second operand is not evaluated.

As before the first operand (`x && y`) of the expression is equal to 0 and according to the quote the second operand (`z++`) is not evaluated.

As result z will be equal to 5 as before.

#177 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{

    int i = 1;

    if (i++ && (i == 1))

        printf("Yes\n");

    else

        printf("No\n");

}
```

- A Yes
- B No
- C compile time error
- D depends on compiler

Correct Answer :B

Explanation

When `&&` is used in an expression, its arguments are guaranteed to be evaluated from left to right.

So `i` will have the value of `2` when `(i==1)` is evaluated.

Therefore, the expression is false, and the `else` part will be executed.

#178 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int a = 10, b = 5, c = 5;
    int d;
    d = b + c == a;
    printf("%d", d);
}
```

- A
0
- B
1

- C
6
- D
5

Correct Answer :B

Explanation

```
#include <stdio.h>

int main()

{
    int a = 10, b = 5, c = 5;

    int d;

    d = b + c == a;

    // Here it will add b + c = 10 then check it equals to 10 that's match the
    // condition and return 1 that store in variable d
```

```
printf("%d", d);
```

#179 Explained Report Bookmark

Which among the following is NOT a logical or relational operator?

- A
!=

- **B**
==
- **C**
||
- **D**
=

Correct Answer :D

Explanation

No Explanation is required here, its a straight forward question

#180 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h> //Program 1

int main()

{

    int d, a = 1, b = 2;

    d = a++ + ++b;

    printf("%d %d %d", d, a, b);

}
```

- **A**
4 2 3

- **B**
3 2 3
- **C**
4 2 2
- **D**
4 2 4

Correct Answer :A

Explanation

```
#include <stdio.h> //Program 1

int main()

{

    int d, a = 1, b = 2;

    d = a++ + ++b;

    // post increment so a = 1 then update its value to 2

    //pre increments of b = 3

    //a++ + ++b = 1 + 3 = 4

    printf("%d %d %d", d, a, b);
```

```
//final output will be 4 2 3
```

}

#181 Explained Report Bookmark

What will be the output of the following C code

```
#include <stdio.h>

void main()
{
    int x = 4, y, z;
    y = --x;
    z = x--;
    printf("%d%d%d", x, y, z);
}
```

- A
3 2 3
 - B
2 3 3
 - C
3 2 2

- D
2 3 4

Correct Answer :B

Explanation

```
#include <stdio.h>

void main()
{
    int x = 4, y, z;

    y = --x;

    // pre decrement so first value will evaluate then
    assign to y = 3

    z = x--;
    //post decrement so first assign value to z then
    decrement value of x = 2

    printf("%d%d%d", x, y, z);

    //final output will be 2 3 3
}
```

#182 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

void main()

{

    int x = 4;

    int *p = &x;

    int *k = p++;

    int r = p - k;

    printf("%d", r);

}
```

- **A**
4
- **B**
1
- **C**
8
- **D**
Run time error

Correct Answer :B

Explanation

Initially p contains address of x. Then address of x is stored in k and p will point to next address due to post increment operator in p++. Therefore p-k gives 1 as the output.

#183 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

void main()
{
    int a = 5, b = -7, c = 0, d;
    d = ++a && ++b || ++c;
    printf("\n%d%d%d", a, b, c, d);
}
```

- **A**
6 -6 0 0
- **B**
6 -5 0 1
- **C**
-6 -6 0 1
- **D**
6 -6 0 1

Correct Answer :D

Explanation

```
#include <stdio.h>

void main()
{
    int a = 5, b = -7, c = 0, d;

    d = ++a && ++b || ++c;

    //++a && ++b  value a is pre-increment so updated
    value is

    //now ++b will increment by 1 as post -increment

    //so 6 && -6 will result as false so next condition
    will executed

    //which will lead to increment of ++c so c has 1

    //++a a = 6

    //++b b = -6

    //++c c = 1

    //d holds true(1)

    printf("\n%d%d%d", a, b, c, d);

    //finla output will be 6 -6 0 1
}
```

#184 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int c = 2 ^ 3;
    printf("%d\n", c);
}
```

- A
1
- B
8
- C
9
- D
0

Correct Answer :A

Explanation

In this program we use Bitwise XOR which will give 1 when either A is 1 or B is 1 , but when both are 1 then the output is 0

First we write 2 and 3 into binary form and it take two bits at a time and then perform Bitwise XOR that number

2 -- 0 0 1 0

3 -- 0 0 1 1

^

0 0 0 1 --- 1

And then it prints the value of c which is 1.

#185 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    unsigned int a = 10;

    a = ~a;

    printf("%d\n", a);
}
```

- A
-9

- **B**
-10
- **C**
-11
- **D**
10

Correct Answer :C

Explanation

Given an 8-bit integer, you flip *all* of the bits, including the leading zeros. In other words, 10 is actually 00001010, not just 1010. Flip them and you get 11110101, which is -11

#186 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int a = 2;

    if (a >> 1)

        printf("%d\n", a);

}
```

- A
0
- B
1
- C
2
- D
no output

Correct Answer :C

Explanation

The operator `<<` is a bitwise left-shift operator.

So when you write `1<<2`, the binary representation of `1` is shifted left by `2` bits

so if condition satisfy and simply print `a = 2`

#187 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
void main()
```

```
{
```

```
    int x = 97;
```

```
int y = sizeof(x++);
printf("x is %d", x);
}

• A
  x is 97
• B
  x is 98
• C
  x is 99
• D
  Run time error
```

Correct Answer :A

Explanation

According to C99 Standards, the sizeof() operator only takes into account the type of the operand, which may be an expression or the name of a type (i.e int, double, float etc) and not the value obtained on evaluating the expression. Hence, the operand inside the sizeof() operator is not evaluated. It is evaluated only if the type of the operand is variable length array because in that case, the size can be determined only after the expression is evaluated.

so final out has x = 97

#188 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int main()
{
    int a = 1, b = 2;
    a += b -= a;
    printf("%d %d", a, b);
}
```

- **A**
1 1
- **B**
1 2
- **C**
2 1
- **D**
2 2

Correct Answer :C

Explanation

```
#include <stdio.h>

int main()
{
    int a = 1, b = 2;
    a += b -= a;
```

```
//In most high-level languages, expressions like this  
are evaluated left to right  
  
//but += is right-to-left associative so  
  
//b = 2 - 1 = 1  
  
//a = 1 + 1 = 2  
  
printf("%d %d", a, b);  
  
//final output will be 2 1  
  
}
```

#189 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>  
  
void main()  
  
{  
  
    int k = 8;  
  
    int m = 7;  
  
    k < m ? k++ : m = k;  
  
    printf("%d", k);  
  
}
```

- A
7
- B
8
- C
compile time error
- D
run time error

Correct Answer :C

Explanation

Ternary operator produces some result, it never assign values inside operation. It is same as a function which has return type. So there should be something to be assigned but unlike inside operator

so here we got

lvalue required as left operand of assignment error

#190 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
void main()
```

```
{
```

```
    int k = 8;
```

```
    int m = 7;
```

```
k < m ? k = k + 1 : m = m + 1;  
  
printf("%d", k);  
  
}
```

- **A**
Compile time error
- **B**
9
- **C**
8
- **D**
run time error

Correct Answer :A

Explanation

lvalue required as left operand of assignment

#191 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>  
  
void main()  
  
{
```

```
1 < 2 ? return 1: return 2;  
}
```

- A returns 1
- B returns 2
- C run time error
- D Compile time error

Correct Answer :D

Explanation

`?:` is an operator not a control flow construct, so the whole thing with operands must be an expression

, and return statements (or any statement) are not valid sub-expressions.

#192 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C function?

```
#include <stdio.h>  
  
int main()  
{
```

```

    reverse(1);

}

void reverse(int i)

{
    if (i > 5)
        exit(0);

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

    return reverse(i++);
}

```

- **A**
1 2 3 4 5
- **B**
1 2 3 4
- **C**
Compile time error
- **D**
segmentation fault

Correct Answer :D

Explanation

We call the main method again and again by 1 because we use post-increment. At certain time stack frame will full means segmentation fault occurs.

#193 Explained Report Bookmark

What will be the output of the following C function?

```
#include <stdio.h>

void reverse(int i);

int main()

{

    reverse(1);

}

void reverse(int i)

{

    if (i > 5)

        return ;

    printf("%d ", i);

    return reverse((i++, i));

}
```

- **A**
1 2 3 4 5
- **B**
Segmentation fault

- **C**
Compilation error
- **D**
Undefined behaviour

Correct Answer :A

Explanation

```
#include <stdio.h>

void reverse(int i);

int main()

{

    reverse(1);

}

void reverse(int i)

{

    if (i > 5)

        return ;

    printf("%d ", i);

    return reverse((i++, i));
}
```

```

        // in first time value of i is 1

        //it simply print 1

        //then we call this function again with (i++, i)

        //brackets has left to right ASSOCIATIVITY so

        // i value incremented by 1 and pass to function

        // this process repeat 5 time

        //so final output will be : 1 2 3 4 5

    }

```

#194[Explained](#) [Report](#) [Bookmark](#)

What will be the final values of i and j in the following C code?

```

#include <stdio.h>

int x = 0;

int main()

{

    int i = (f() + g()) || g();

    int j = g() || (f() + g());

```

```
}
```



```
int f()
```



```
{
```



```
    if (x == 0)
```



```
        return x + 1;
```



```
    else
```



```
        return x - 1;
```



```
}
```

```
int g()
```



```
{
```



```
    return x++;
```



```
}
```

- **A**
i value is 1 and j value is 1
- **B**
i value is 0 and j value is 0
- **C**
i value is 1 and j value is undefined
- **D**
i and j value are undefined

Correct Answer :D

Explanation

int i = (f() + g()) || g(); has an compilation error

so value of i and j will be undefined

#195 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int x = 2, y = 0;

    int z = (y++) ? 2 : y == 1 && x;
    printf("%d\n", z);

    return 0;
}
```

- A
0
- B
1
- C
2

- D
error

Correct Answer :B

Explanation

```
int x = 2, y = 0;  
  
int z = (y++) ? 2 : y == 1 && x;
```

The post-increment runs first (due to the braces; even without them braces it should 'win' since it also has the highest precedence amongst the ones in the expression) and it returns the value of the operand `y` first and then does the increment hence it becomes

```
int z = 0 ? 2 : 1 == 1 && 2;
```

Now the operators involved are

- `=` (assignment)
- `?:` (ternary condition)
- `==` (equality)
- `&&` (logical and)

Thus is the precedence among them: `== > && > ?: > =`. Thus the stages of evaluation of the expression becomes

```
int z = 0 ? 2 : true && 2;      // == evaluated to true
```

```
int z = 0 ? 2 : true;           // 2 in boolean context evals to
2; true && true = true
```

```
int z = true;                  // ?: chooses the 2nd operand
since the condition is false
```

```
int z = 1;                      // true implicitly converted to
int gives 1
```

Thus you see `z` being printed as `1`.

#196 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int x = 2, y = 0;

    int z;
    z = (y++, y);

    printf("%d\n", z);
```

```
    return 0;  
}  
  
• A  
0  
• B  
1  
• C  
2  
• D  
error
```

Correct Answer :B

Explanation

```
#include <stdio.h>  
  
int main()  
{  
  
    int x = 2, y = 0;  
  
    int z;  
  
    z = (y++, y);  
  
    // () has left to right ASSOCIATIVITY  
  
    // so first y++ increment then assign to z
```

```
// z = 1

printf("%d\n", z);

//final output will be print as 1

return 0;

}
```

#197 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

void main()

{

    int b = 5 - 4 + 2 * 5;

    printf("%d", b);

}
```

- **A**
25
- **B**
-5
- **C**
11

- D
16

Correct Answer :C

Explanation

```
#include <stdio.h>

void main()
{
    int b = 5 - 4 + 2 * 5;
    // * has highest precedence
    //so 2 * 5 = 10
    //then 5 - 4 = 1
    //in final output 1 + 10 = 11
    printf("%d", b);
}
```

#198 Explained Report Bookmark

What will be the output of the following C code?

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

```
void main()  
  
{  
  
    int b = 5 & 4 & 6;  
  
    printf("%d", b);  
  
}
```

- **A**
5
- **B**
6
- **C**
3
- **D**
4

Correct Answer :D

Explanation

: In this program We have to perform Bitwise AND which give result 1 when both bits are 1 and IT take 2 bits at a time

5 -- 0 1 0 1

4 – 0 1 0 0

&

0 1 0 0 which is 4 in decimal, now we perform again Bitwise AND (4 & 6)

4 -- 0 1 0 0

6 -- 0 1 1 0

&

0 1 0 0 which is 4 in decimal and this 4 will assign in variable b and we get 4 as our output

#199 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
#include<stdio.h>

main()

{
    for(;;)printf("Hello");
}
```

- A
Infinite loop
- B
Prints “Hello” once.

- **C**
No output
- **D**
Compile error

Correct Answer :A

Explanation

infinite loop, with second expression of ‘for’ being absent it is considered as true by default.

#200 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
main()
```

```
{
```

```
    union abc {
```

```
        int x;
```

```
        char ch;
```

```
    }var;
```

```
var.ch = 'A';  
  
printf("%d", var.x);  
  
}
```

- **A**
A
- **B**
garbage value
- **C**
65
- **D**
97

Correct Answer :C

Explanation

65, as the union variables share common memory for all its elements, x gets 'A' whose ASCII value is 65 and is printed.

#201 [Explained](#) [Report](#) [Bookmark](#)

The default executable generation on UNIX for a C program is ____

- **A**
a.exe
- **B**
q
- **C**
a.out
- **D**
out.a

Correct Answer :C

Explanation

"a.out" is the default name of the executable generated on both the UNIX and Linux operating systems.

#202 [Explained](#) [Report](#) [Bookmark](#)

Where to place "f" with a double constant 3.14 to specify it as a float?

null

- A
(float)(3.14)(f)
- B
(f)(3.14)
- C
3.14f
- D
f(3.14)

Correct Answer :C

Explanation

A floating-point constant without an f, F, l, or L suffix has type double. If the letter f or F is the suffix, the constant has type float. If suffixed by the letter l or L, it has type long double.

#203 [Explained](#) [Report](#) [Bookmark](#)

According to ANSI specification, how to declare main () function with command-line arguments?

null

- **A**
int main(int argc, char *argv[])
- **B**
int char main(int argc, *argv)

C

int main()

{

Int char (*argv argc);

-)
- **D**
None of the above

Correct Answer :A

Explanation

Some time, it becomes necessary to deliver command line values to the C programming to execute the particular code when the code of the program is controlled from outside. Those command line values are called command line arguments. The command line arguments are handled by the main() function.

Declaration of main () with command-line argument is,

```
int main(int argc, char *argv[])
```

Where, argc refers to the number of arguments passed, and argv[] is a pointer array which points to each argument passed to the program.

#204 Explained Report Bookmark

The operator & is used for

- **A**
Bitwise AND
- **B**
Bitwise OR
- **C**
Logical AND
- **D**
Logical OR

Correct Answer :A

Explanation

No Explanation is required here, its a straight forward question

#205 Explained Report Bookmark

Build-in data structure in C are

- **A**
Array
- **B**
Files
- **C**
Structures
- **D**
All

Correct Answer :A

Explanation

No Explanation is required here, its a straight forward question

#206 [Explained](#) [Report](#) [Bookmark](#)

What is the size of a character variable in C is

- **A**
4 Bytes
- **B**
8 Bytes
- **C**
16 Bytes
- **D**
None of the above

Correct Answer :D

Explanation

No Explanation is required here, its a straight forward question

#207 [Explained](#) [Report](#) [Bookmark](#)

If x is an array of interger, then the value of &x[i] is same as

null

- **A**
&x[i-1] + sizeof (int)
- **B**
x + sizeof (int) * i
- **C**
x+i
- **D**
none

Correct Answer :A

Explanation

$X+i$ means increment in value of X not in address of X so it can't represent address of X .

$\&x[i]$ means address of the i^{th} element.

So $\&X[i-1]$ defines address of $i-1$ element . $\text{sizeof}(\text{int})$ defines size of an element

So $\&x[i-1] + \text{sizeof}(\text{int})$ means address of $i-1$ element plus size of an element that means address of i^{th} element.

It can't be option c as $x+i$ is not representing any address.

$X+i$ means increment in value of X not in address of X so it can't represent address of X .

So option (A) is correct

#208 [Explained](#) [Report](#) [Bookmark](#)

If S is an array of 80 characters, then the value assigned to S through the statement $\text{scanf}("%s", S)$ with input 12345 would be

null

- **A**
"12345"
- **B**
nothing since 12345 is an integer

- **C**
S is an illegal name for string
- **D**
%s cannot be used for reading in values of S

Correct Answer :A

Explanation

scanf("%s", S) only scans first parameter of input stream .So Option A is correct.

#209 [Explained](#) [Report](#) [Bookmark](#)

Size of the array need not be specified, when

null

- **A**
Initialization is a part of definition
- **B**
It is a declaratrion
- **C**
It is a formal parameter
- **D**
All of these

Correct Answer :A

Explanation

Consider the following declaration

```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

If you omit the size of the array, an array just big enough to hold the initialization is created. So, correct option is 'A'.

#210 Explained Report Bookmark

A one dimensional array A has indices 1....75. Each element is a string and takes up three memory words. The array is stored starting at location 1120 decimal. The starting address of A[49] is

null

- **A**
1167
- **B**
1164
- **C**
1264
- **D**
1169

Correct Answer :C

Explanation

One element takes three memory words so memory location 1120 , 1121 , 1122 stores first element.

A[49] will be stored at location 1264 , $(1120 + (48 * 3))$

#211 Explained Report Bookmark

**Minimum number of interchange needed to convert the array
89,19,40,14,17,12,10,2,5,7,11,6,9,70, into a heap with the maximum element at the root is**

null

- A 0
- B 1
- C 2
- D 3

Correct Answer :C

Explanation

Only element 70 violates the rule. Hence, it must be shifted to its proper position.

Step1: swap(10, 70)

Step2: swap(40, 70)

Hence, only 2 interchanges are required.

#212[Explained](#) [Report](#) [Bookmark](#)

Which of the following is an illegal array definition?

- A Type COLOGNE:(LIME,PINE,MUSK,MENTHOL); var a:array[COLOGNE]of REAL;

- **B**
var a:array[REAL]of REAL;
- **C**
var a:array['A'..'Z']of REAL;
- **D**
var a:array[BOOLEAN]of REAL;

Correct Answer :B

Explanation

Array index must be integers. Enumerators, characters and boolean can be used in place of an integer but not real.

Option A: Size 4.(colonge is already defined with 4 values, so we know the size of this array)

Option B: Size uncountable.(nothing range or size is defined .. i mean what is real inside array size..)

Option C: Size 26.(again a....z gives the size of array as total number of alphabets..)

Option D: Size 2.(boolean has only 2 values ... so we know the size of array)

#213 Explained Report Bookmark

Minimum number of comparison required to compute the largest and second largest element in array is

- A $n - \lceil \log_2 n \rceil - 2$
- B $n + \lceil \log_2 n - 2 \rceil$
- C $\log_2 n$
- D None of these

Correct Answer :B

Explanation

we can start comparing the elements pair wise and build a decision tree in which case it takes $n-1$ comparisons to obtain the highest element.

In order to get the second highest element we need to check only those elements which were compared with the highest element while building the decision tree, now as the height of the tree is $\log n$ as $\log n$ comparisons are required for the highest element to reach the top, so to obtain second highest element one need $\log n - 1$ comparisons so all together $n-1 + \log n - 1 = n + \log n - 2$ comparisons are required

#214 Explained Report Bookmark

If storage class is missing in the array definition, by default it will be taken to be

null

- **A**
automatic
- **B**
external
- **C**
static
- **D**
either automatic or external depending on the place of occurrence

Correct Answer :D

Explanation

If it is coming within a function, the storage class will be taken to be automatic. otherwise external

#215 [Explained](#) [Report](#) [Bookmark](#)

Consider the array definition

int num [10] = {3, 3, 3};

- **A**
num [9] is the last element of the array num
- **B**
The value of num [8] is 3
- **C**
The value of num [3] is 3
- **D**
None of the above

Correct Answer :A

Explanation

In array it always start it's indexing from 0

So the num[9] is the last element of the array num

#216 Explained Report Bookmark

Consider the following type definition.

```
typedef char x[10];
```

```
x myArray[5];
```

What will sizeof(myArray) be ? (Assume one character occupies 1 byte)

- A 15 bytes
- B 10 bytes
- C 50 bytes
- D 30 bytes

Correct Answer :C

Explanation

`typedef char x[10];` defines a new type `x` as an array of 10 `char`s. So, `myArray` is an array of 5 `x`--which is an array of 10 `char`s.

That is `myArray` is actually of `char [5] [10]` type; an array of 5 elements of which each element is an array of 10 `char`s.

Therefore `sizeof (myArray)` will return 50.

#217 Explained Report Bookmark

While passing an array as an actual argument, the function call must have the array name

- A with its size
- B alone
- C none of the above
- D with empty brackets

Correct Answer :B

Explanation

While passing an array as an actual argument, the function call must have the array name alone.

An array reference always "decays" to a pointer to the first element of the array. Therefore, it isn't possible to pass an array "by value". An array in a function call will be passed to the function as a pointer, which is analogous to passing the array by reference.

While passing arrays as arguments to the function, only the name of the array is passed. C program to pass an array containing age of person to a

function. This function should find average age and display the average age in main function

#218 [Explained](#) [Report](#) [Bookmark](#)

The following program

```
main( )  
  
{  
  
    static int a[ ] = { 7, 8, 9 } ;  
  
    printf( "%d", 2[ a ] + a[ 2 ] ) ;  
  
}
```

- **A**
results in bus error
- **B**
results in segmentation violation error
- **C**
will not compile successfully
- **D**
none of the above

Correct Answer :D

Explanation

a[2] will be converted to *(a + 2).

*(a + 2) can as well be written as *(2 + a) .

$*(2 + a)$ is nothing but $2[a]$. So. $a[2]$ is essentially same as $2[a]$ which is same as $*(2 + a)$. So. it prints $9 + 9 = 18$.

Some of the modern compilers don't accept $2[a]$.

#219 [Explained](#) [Report](#) [Bookmark](#)

The parameter passing mechanism for an array is

- A call by value
- B call by value-result
- C call by reference
- D none of these

Correct Answer :A

Explanation

Parameter passing mechanism for an array is call by value. This mechanism is used for passing objects, where a reference to the object is passed by value.

#220 [Explained](#) [Report](#) [Bookmark](#)

Under which of the following conditions, the size of an one-dimensional array need not be specified?

- A when initialization is a part of definition
- B when it is a declaration

- **C**
when it is a formal parameter and an actual argument
- **D**
All of the above

Correct Answer :D

Explanation

All, size is not specified when it come to one dimensional array

#221 [Explained](#) [Report](#) [Bookmark](#)

If n has the value 3, then the statement `a[++n]=n++;`

- **A**
assigns 3 to a [5]
- **B**
assigns 4 to a [5]
- **C**
assigns 4 to a [4]
- **D**
what is assigned is compiler-dependent

Correct Answer :D

Explanation

Look, here:

for the variable "n". There are two things:

1) Use "n" as the index of the array a.(After doing `++n`)

2) Increment the the value of "n" by 1.(by doing `n++`)

Now, the confusion is :

whether to use the value of "n" before it is modified (incremented by doing `n++`) as the index, or after it . Different compilers will try to solve this confusion, by doing something like:

a) doing `++n` on the value of "n" first (to be used as the index)

b) doing `n++` first, and then `++n` to calculate the index

This is undefined behaviour, and will depend on how a particular compiler handles such situation.

#222Explained Report Bookmark

Choose the correct statements

- A
Array stores data of the same type
- B
Array can be a part of a structure
- C
Array of structure is allowed
- D
All of the above

Correct Answer :D

Explanation

A) An array is a homogeneous data structure (elements have same data type) (TRUE)

B)we can declare an array without a dimension and whose size is flexible in nature. Such an array inside the structure should preferably be declared as the last member of structure and its size is variable(can be changed be at runtime) (TRUE)

C)An array of structures is simply an array in which each element is a structure of the same type. The referencing and subscripting of these arrays (also called structure arrays) follow the same rules as simple arrays.(TRUE)

#223 [Explained](#) [Report](#) [Bookmark](#)

Consider the declaration

```
char street[10] = "abcdefghi";
```

Choose the correct remark(s)

- A &street and street will have different values
- B &street is meaningless
- C &street+1 and street+1 will have the same values
- D None of the above

Correct Answer :D

Explanation

`&street` and `street` will have the values which is the starting address of the `street` array. However, `street` is a pointer to the first character whereas `&street` is a pointer to the entire array. The incremented values of `street` and `&street` reflects this difference.

#224 [Explained](#) [Report](#) [Bookmark](#)

For loop in a C program, if the condition is missing

- A it is assumed to be present and taken to be false
- B it is assumed to be present and taken to be true
- C it result in a syntax error
- D execution will be terminated abruptly

Correct Answer :B

Explanation

In for loop if condition is missing it will assume true all time and execute the loop infinite time

#225 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statement about for loop is true ?

- A Index value is retained outside the loop
- B Index value can be changed from within the loop
- C Goto can be used to jump, out of the loop

- **D**
All of these

Correct Answer :D

Explanation

In for loop , you can use goto statements to jump out of the loop . It can be used to jump anywhere in the program

And within the loop the value of the index changes. You can increment or decrement the value according to the loop condition . and the value of the index is retained outside the loop .

#226[Explained](#) [Report](#) [Bookmark](#)

If c is a variable initialised to 1, how many times will the following loop be executed?

```
while ((c > 0) && (c < 60))
```

```
{
```

```
    loop body
```

```
    c ++;
```

```
}
```

- **A**
60
- **B**
59

- **C**
61
- **D**
None of these

Correct Answer :B

Explanation

In the while loop when both conditions are true then it enters inside the loop and the condition is c is greater than zero and c is less than 60 ,and we initialize the c is 1. So the loop will execute 59 times and when it checks the condition at 60th time the second condition becomes false and it comes out of the loop

#227Explained Report Bookmark

How many times will the following loop be executed if the input data item is 0 1 2 3 4 ?

```
while (c = getchar () ! = 0)  
{ }
```

- **A**
Ininitely
- **B**
never
- **C**
once
- **D**
None of these

Correct Answer :A

Explanation

because in this program !=(not equal to) has higher precedence than =(assignment operator)

This does not give output but get execute infinitely

#228 [Explained](#) [Report](#) [Bookmark](#)

Consider following program fragment char c ='a' ;

while (c++ <= 'z')

putchar (xxx) ;

If required output is abcd....xyz, then xxx should be

- A
c++
- B
c-2
- C
c-1
- D
--c

Correct Answer :C

Explanation

In while loop we get the condition (c++ <= 'z')

And char c is assign with value 'a' and the ASCII value of a is 97 and z = 122

When we start checking the condition we get post increment which means first use then increment the value by 1 means $(97 \leq 122)$, which is true

And then it comes inside the loop here we get putchar and if we use c-1 in it . It will give desired output . because as we get post increment in our condition so while printing the character from a to z we have to reduce it by 1

Then we get our output as “abc.....xyz”

#229 [Explained](#) [Report](#) [Bookmark](#)

A "switch" statement is used to

- **A**
Switch between functions in a program
- **B**
Switch from one variable to another variable
- **C**
To choose from multiple possibilities which may arise due to different values of a single variable
- **D**
All of above

Correct Answer :C

Explanation

Switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed. Each case in a block of a switch has a different name/number which is referred to as an identifie

#230 [Explained](#) [Report](#) [Bookmark](#)

What is the output of given program if user enter "xyz" ?

```
#include <stdio.h>

void main()
{
    float age, AgeInSeconds;
    int value;

    printf("Enter your age:");
    value=scanf("%f", &age);

    if(value==0) {
        printf("\nYour age is not valid");
    }

    AgeInSeconds = 365 * 24 * 60 * 60 * age;

    printf("\n You have lived for %f seconds",
AgeInSeconds);
}
```

- **A**

Enter your age : xyz Your age is not valid

- **B**
Enter your age: xyz You have lived for 0 seconds
- **C**
Enter your age: xyz You have lived for 0.00 seconds
- **D**
Compiler error

Correct Answer :D

Explanation

When we give scanf() a "%f" format string, that means "We want you to try and get us a floating point number. When we provide input like 'xyz', it's not going to match anything, because 'xyz' is not a valid floating-point number.

#231 [Explained](#) [Report](#) [Bookmark](#)

Choose the correct statement

- **A**
use of goto enhances the logical clarity of the code
- **B**
use of goto makes the debugging task easier
- **C**
use goto when you want to jump out of a nested loop
- **D**
never use goto

Correct Answer :C

Explanation

goto statement is also called a jump statement , which is used to jump anywhere in the program and with the help of goto statement we can come out of the loop.

#232 [Explained](#) [Report](#) [Bookmark](#)

Choose the statements that are syntactically correct

- **A**
/* Is /* this a valid */ comment */
- **B**
for (; ;);
- **C**
return;
- **D**
both (b) & (c)

Correct Answer :D

Explanation

In programming we can write return; like this which mean it return nothing which is use when the function is void type

The syntax of for loop

for(Initialization ; condition ; increment/decrement)

{

Statements;

}

And this for(;;); In this for loop Initialization , condition and increment /decrement is undefined , so the control will remain in the loop indefinitely but it will never enter inside the loop because we use termination sign (; semicolon) after the loop , which terminate the loop

#233 [Explained](#) [Report](#) [Bookmark](#)

The following program fragment

```
for(i = 3 ; i < 15; i += 3);  
printf(" %d ", i);
```

- **A**
a syntax error
- **B**
an execution error
- **C**
printing of 12
- **D**
printing of 15

Correct Answer :D

Explanation

:: In this for loop we get termination sign so the loop run like this first initialize , second check condition but because of termination sign it never come inside the loop so it execute the increment part and when condition is false it print the value of i which is 15

i=3 , then $3 < 15$ which is true and then it execute $i += 3$ means $i = i + 3 / 3 + 3 = 6$

now $i = 6$, $6 < 15$ (true) then $6 + 3 = 9$

now i=9, $9 < 15$ (true) then $9+3 = 12$

now i=12 , $12 < 15$ (true) then $12+3 = 15$

now i=15 , $15 < 15$ (condition false) , come out of the loop

and print 15.

#234 Explained Report Bookmark

The following program fragment

```
for (i = 1; i< 5; ++ i )  
  
if ( i == 3) continue;  
  
else printf( " %d " i );
```

- A
1 2 4 5
- B
1 2 4
- C
2 4 5
- D
none of the above

Correct Answer :B

Explanation

: In this program $i = 1$, then it check the condition $1 < 5$ which is true then it enter inside the loop and check if condition $1 == 3$ which is false, and execute the else statement and print 1 on console

After executing it increment the value by 1

Now i=2 , 2<5 (true) , check if condition 2==3 (false), execute else statement and print 2 and then increment value by 1

Now i=3 , 3<5 (true) , check if condition 3==3 (true), execute if statement and continue the loop and then increment value by 1

Now i=4 , 4<5 (true) , check if condition 4==3 (false), execute else statement and print 4 and then increment value by 1

Now i=5 , 5<5 (condition false) and then it come out of the loop

#235 [Explained](#) [Report](#) [Bookmark](#)

The following loop

```
for ( putchar( 'c' ) ; putchar ( 'a' ) ; putchar ( ' r' ) )  
    putchar ( 't' );
```

- **A**
a syntax error
- **B**
cartrt
- **C**
catrat
- **D**
catratratratrat

Correct Answer :D

Explanation

As per for loop execute

for(1. Initialization;2.Condition Check; 3. Iteration Variable change)

Now as per given question

1. Initialization is given by putchar('c')

2. Condition Check by putchar('a')

3. Iteration by putchar('r')

In for loop statement putchar('t') is there

Also sequence is

1. Initialization only once putchar('c')

2. Condition check putchar('a')

3. Statement putchar('t')

4. Iteration putchar('r')

2. putchar('a')

3. putchar('t')

4. putchar('r')

2. putchar('a')

3. putchar('t')

.

and so on so result will catratratrat.... Infinite times

#236 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
for( i=1, j=10; i<6; ++i, --j )
```

```
printf ("%d %d", i, j );
```

- A
1 10 2 9 3 8 4 7 5 6
- B
1 2 3 4 5 10 9 8 7 6
- C
1 1 1 1 1 1 9 9 9 9 9
- D
none of above

Correct Answer :A

Explanation

In this program we get two variables at the place of initialization and increment decrement . but execution of loop will happens same

First it will initialize the variables then check the condition whether it is true or false

And if condition is true it comes inside the loop and execute the printf statement

In this program it start with initialization of i and j

I = 1 and j =10 , then it check condition $1 < 6$ (true), come inside the loop and print value of i and j , then it increment and decrement the values

Now i = 2 and j = 9 , $2 < 6$ (true) print values of i and j , and then execute increment and decrement.

Now i = 3 and j = 8 , $3 < 6$ (true) print values of i and j , and then execute increment and decrement.

Now i = 4 and j = 7 , $4 < 6$ (true) print values of i and j , and then execute increment and decrement.

Now i = 5 and j = 6 , $5 < 6$ (true) print values of i and j , and then execute increment and decrement.

Now i= 6 and j= 5 , $6 < 6$ condition false and it come out of the loop

#237 Explained Report Bookmark

What is the output of the below code snippet?

```
main()
{
    int i = 5;
    if (i == 5) return;
    else printf (" is not five");
    printf("over");
}
```

- **A**
a syntax error
- **B**
an execution error
- **C**
printing of over
- **D**
execution termination, without printing anything

Correct Answer :D

Explanation

In this program no output will print on console because $i = 5$ and condition gets true and we come inside the if statement . so it return nothing because we write return only (without any value)

#238 Explained Report Bookmark

In a for loop, if the condition is missing, then,

- A it is assumed to be present and taken to be false
- B it is assumed to be present and taken to be true
- C it results in a syntax error
- D execution will be terminated abruptly

Correct Answer :B

Explanation

In for loop if condition is missing it will assume true all time and execute the loop infinite time

#239 Explained Report Bookmark

If switch feature is used, then

- A default case must be present
- B default case, if used, should be the last case
- C default case, if used, can be placed anywhere
- D none of the above

Correct Answer :C

Explanation

The default case can be placed anywhere in the switch case. Even if we don't include the default case, switch statement works.

#240 [Explained](#) [Report](#) [Bookmark](#)

What does a preprocessor?

- **A**
Compiles a program
- **B**
It loads a program into memory for execution
- **C**
It links all the required modules and files to the program
- **D**
It provides an expand source code to the compiler

Correct Answer :D

Explanation

preprocessor will expand the code It is the separate step of the compilation step because it is somewhere written , we just include in the program according to the need

Eg #include include the other functionality in the program

#241 [Explained](#) [Report](#) [Bookmark](#)

The reason for discouraging macros in C is

- **A**
It slows down execution of executable code
- **B**
It makes code difficult to understand
- **C**
It increases memory requirements
- **D**
It expands the source code for compiler

Correct Answer :C

Explanation

: It is difficult to understand the macros . macro is used for constant values . they are not like variable which we can change in our program

#242[Explained](#) [Report](#) [Bookmark](#)

#define is _____.

- **A**
a preprocessor
- **B**
a macro itself
- **C**
a preprocessor directive
- **D**
a template

Correct Answer :C

Explanation

#define is the c preprocessor directive used to define macros and allow the definition of macro in a program . Macros are not like variable and we cannot change in our program

It is used to allow constant values in the program

#243[Explained](#) [Report](#) [Bookmark](#)

Which of the following statement is true regarding macros and functions?

- **A**
Macros and functions are same
- **B**
Macros run program faster than functions
- **C**
Due to functions the program size increases, hence macros are preferred
- **D**
Macros increases overheads of an activation record

Correct Answer :B

Explanation

Speed of execution using Macro is faster and speed of execution using function is slower.

#244[Explained](#) [Report](#) [Bookmark](#)

Which of the following statement is Incorrect?

- **A**
Macros does not have address
- **B**
Macros does not perform data type checking

- **C**
Macros are handled by preprocessor
- **D**
Macros can not be redeclared in same file

Correct Answer :D

Explanation

Macros can be redeclared in same file after becoming undefined using #undefmacro_name or we can say a macro can be redefined after it has been removed by the #undef directive.

#245 Explained Report Bookmark

What is the Correct execution sequence?

- **A**
Compiler => Preprocessor => Linker
- **B**
Compiler =>Linker => Preprocessor
- **C**
Preprocessor =>Compiler =>Linker
- **D**
Preprocessor =>;Linker =>Compiler

Correct Answer :C

Explanation

Before compilation, preprocessor directives are resolved by preprocessor through copying files indicated by preprocessor directives in the source program. After all preprocessor directives get resolved, the source code is

compiled with the help of the compiler and after compilation linker links required library files with the source code.

#246 [Explained](#) [Report](#) [Bookmark](#)

Which header file is not available in standard library?

- A string.h
- B ctype.h
- C matrix.h
- D time.h

Correct Answer :C

Explanation

matrix.h header file is not available in standard library.

#247 [Explained](#) [Report](#) [Bookmark](#)

Which is not a preprocessor directive?

- A include
- B undef
- C pragma
- D elifdef

Correct Answer :D

Explanation

elifdef is not a preprocessor directive in C programming language

#248 [Explained](#) [Report](#) [Bookmark](#)

Macro can be disabled using preprocessor directive_____.

- A
undefine
- B
undef
- C
dontdef
- D
undefined

Correct Answer :B

Explanation

In the C programming language, the #undef directive tells the preprocessor to remove all definitions for the specified macro. A macro can be redefined after it has been removed by the #undef directive. Once a macro is undefined, an #ifdef directive on that macro will evaluate to false.

#249 [Explained](#) [Report](#) [Bookmark](#)

Macro definition can be extended on more than one lines using character_____.

- A
/

- **B**
\
- **C**
#
- **D**
%

Correct Answer :B

Explanation

Macro definition can be extended on more than one line using the backslash character ('\'). The backslash character is the line continuation character. It must be the last character on a line. The preprocessor joins lines ending with a line continuation character into a single line (for purpose of preprocessing and compilation). The last line doesn't need to have '\'.

#250Explained Report Bookmark

What will be the output of the following C code?

```
# define square(x) (x*x)
```

```
void main( )
```

```
{
```

```
    int a,b=3;
```

```
    a=square(b+2);
```

```
    printf("%d", a);
```

```
}
```

- A
25
- B
9
- C
11
- D
garbage

Correct Answer :C

Explanation

x will be replaced with b+2. Before compilation, square(b+2) will be changed to b+2*b+2 and value of b is 3 and finally 3+2*3+2 will be equal to 11.

#251 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#define calc(a) (a*a*a);

void main( )

{
    float i=2.0,p=0;

    p=calc(i);

    printf("%f", p);

    if(calc(i)==8) printf("cube");
}
```

}

- **A**
8
- **B**
8.000000cube
- **C**
8.000000
- **D**
compilation error

Correct Answer :D

Explanation

Compilation error will occur due to semicolon after #define calc(a) (a*a*a).

#252 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#define square(x, y) (x+y)
```

```
void main( )
```

```
{
```

```
    int i, a=4, b=3;
```

```
    i=add(a+4, 1+b);
```

```
    printf("%d", i);
```

```
}
```

- A
14
- B
13
- C
12
- D
Compilation error

Correct Answer :D

Explanation

error : undefined reference to `add' as we declared macro with square name o it will throw an error

#253 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#define result(a) (a+2*3)
```

```
void main( )
```

```
{
```

```
int x=3,y,z;
```

```

y=result(x++) ;

z=result(++x) ;

printf("%d %d", y, z) ;

}

```

- **A**
9 11
- **B**
15 18
- **C**
15 11
- **D**
9 18

Correct Answer :A

Explanation

result(x++) means in place of a, x++ will be substituted and result(++x) means in place a, x++ will be substituted. Before compilation, y=result(x++) will be changed to y=x++ +2*3 and z=result(++x) will be changed to z=++x+2*3.

#254 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#define TANK(a) a*10+2

int main()

```

```
{  
  
    int a = TANK(2)*2;  
  
    printf("%d", a);  
  
    return 0;  
  
}
```

- **A**
24
- **B**
44
- **C**
22
- **D**
Error

Correct Answer :A

Explanation

TANK(2)*2 is replaced by $2*10+2*2$.

It gives only 24.

Remember, you should have put $(a*10+2)$ parenthesis TANK definition to get expected results.

#255 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#define D

void main( )

{
    int i=2;

#ifdef D

printf("%d",i*i);

#else printf("%d",i);

#endif

}

```

- **A**
2
- **B**
4
- **C**
16
- **D**
Compilation error

Correct Answer :B

Explanation

D has been #defined. If D has not been defined as a macro, printf("%d",i*i) won't be executed but printf("%d",i) will be executed.

#256 Explained Report Bookmark

What will be the output of the following C code?

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

```
#define IMP
```

```
#ifdef IMP
```

```
    int = 10;
```

```
#endif;
```

```
void main( )
```

```
{
```

```
    int = 5;
```

```
    printf("%d", IMP);
```

```
}
```

- A
10
- B
0
- C
5
- D
Error

Correct Answer :D

Explanation

```
#include <stdio.h>

#include <stdio.h>

#define IMP

#ifndef IMP

    int = 10; // Error 1 : no identifier

#endif; // Error 2 : semicolon is not required

void main( )

{

    int = 5; // Error 3 : no identifier

    printf("%d", IMP); // Error 4 : Expected expression

}
```

#257 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

#define mult(x) x*x

int mul(x) {return x*x; }
```

```
void main( )  
{  
    int m,i=3; m=mul(i+2);  
  
    printf("%d",m); m=mul(2+i);  
  
    printf("%d",m);  
}
```

- **A**
1125
- **B**
2525
- **C**
117
- **D**
257

Correct Answer :B

Explanation

2525 will be printed on the screen due to two function calls have been made. Function calls `mul(i+2)` and `mul(2+i)` have been made one after other.

#258 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#define CASE (a>=65&& a<=90)
```

```
void main( )  
{  
    char b='Q';  
  
    if(CASE)  
  
        printf("Upper case");  
  
    else  
  
        printf("Lower case");  
}
```

- **A**
Lower case
- **B**
Upper case
- **C**
Compilation error
- **D**
None of these

Correct Answer :C

Explanation

error: 'a' undeclared (first use in this function)

#259 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```

#include <stdio.h>

#define NOW

#define LATER

void main( )

{
    #ifdef NOW && LATER

        printf("We will go for movie");

    #else

        printf("We will not go");

    #endif
}

```

- **A**
We will not go
- **B**
We will go for movie
- **C**
Compilation error
- **D**
none

Correct Answer :B

Explanation

NOW and LATER has been defined so the condition NOW && LATER becomes true and We will go for movie will be printed on the screen. There is no need to write macro expansion after #define NOW or #define LATER.

#260 [Explained](#) [Report](#) [Bookmark](#)

Which of the following C expression is equivalent to the algebraic expression :

$3x^2 + 3(4x+6)$.

- A $3*x*x+3*(4x+6)$
- B $3*x*x+3(4x+6)$
- C $3*x*x+3(4*x+6)$
- D $3*x*x+3*(4*x+6)$

Correct Answer :D

Explanation

Arithmetic operators, unary operators, relational and logical operators, assignment operators and the conditional operator are used to form expressions.

The data items that operators act upon are called operands. Some operators require two operands, while others act upon only one operand. Most operators allow the individual operands to be expressions. A few operators permit only single variables as operands.

Example:

$c = a + b;$

Here, ' $c=a+b$ ' whole is the expression and '+' and '=' are operators.

In C programming language, every operator has to be explicitly written in the expression.

#261 [Explained](#) [Report](#) [Bookmark](#)

In the following expression which operation is performed first?

Expression: $a=3/2+9*6/4-7+2.4/8$

- A
3/2
- B
9*6
- C
6/4
- D
2+9

Correct Answer :A

Explanation

/ and * both have same precedence. When precedence is same for operators then associativity comes into picture. Associativity of / and * is from left to right so $3/2$ will be evaluated first.

#262 [Explained](#) [Report](#) [Bookmark](#)

Which of the following are unary operators?

- **A**
=(assignment)
- **B**
%(modulus)
- **C**
-(subtraction)
- **D**
/ (division)

Correct Answer :C

Explanation

Types of unary operators:

1. unary minus(-)
2. increment(++)
3. decrement(- -)
4. NOT(!)
5. Addressof operator(&)
6. sizeof()

#263 [Explained](#) [Report](#) [Bookmark](#)

How many ternary operators are available in C?

- **A**
1
- **B**
2
- **C**
3

- **D**
Compiler dependent

Correct Answer :A

Explanation

Only 1 Available

. The ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison

#264 [Explained](#) [Report](#) [Bookmark](#)

Which can not be used as assignment operator?

- **A**
%=
• **B**
!=
• **C**
^=
• **D**
&=

Correct Answer :D

Explanation

Assignment operators are used to assign some value to a data object

OPERAT OR	MEANING	EXAMPLE (A = 10 , B = 5)
<code>+=</code>	$L=L+R$ add left and right operand and assign result in left	a+=b; same as a=a+b after execution a will hold 15
<code>-=</code>	$L=L-R$ subtract right operand from left operand and assign result in left	a-=b; same as a=a-b after execution a will hold 5
<code>*=</code>	$L=L*R$ multiply both right and left operand and store result in left	a*=b; same as a=a*b after execution a will hold 50
<code>/=</code>	$L=L/R$ divides left operand by right operand and store result in left	a/=b; same as a=a/b after execution a will hold 2

%= $L=L \% R$	After left and right operand division, the remainder will be stored in left	$a \% = b$; same as $a = a \% b$ after execution a will hold 0
-----------------------------	---	---

#265 Explained Report Bookmark

Which is not a operator in C?

- **A**
^
- **B**
~
- **C**
sizeof
- **D**
\$

Correct Answer :D

Explanation

An operator is a symbol that tells the compiler to perform a certain mathematical or logical manipulation. Operators are used in programs to manipulate data and variables.

\wedge	Binary XOR Operator copies the bit if it is set in one operand but not both.	$(A \wedge B) = 49$, i.e., 0011 0001
----------	--	--

~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e.,
---	--	---------------------

sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
----------	---------------------------------	---

\$ is not an operator in C

#266 [Explained](#) [Report](#) [Bookmark](#)

Find invalid bitwise operator

- A
^
- B
>>
- C
~
- D
&&

Correct Answer :D

Explanation

Bitwise Operators

OPERATOR	DESCRIPTION	EXAMPLE
&	Binary AND Operator copies a bit to the result if it exists in both operands.	$(A \& B) = 12$, i.e., 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	$(A B) = 61$, i.e., 0011 1101
\wedge	Binary XOR Operator copies the bit if it is set in one operand but not both.	$(A \wedge B) = 49$, i.e., 0011 0001
\sim	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	$(\sim A) = \sim(60)$, i.e., 1100 0011
$<<$	Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand.	$A << 2 = 240$ i.e., 1111 0000
$>>$	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	$A >> 2 = 15$ i.e., 0000 1111

Logical Operators

OPERATOR	DESCRIPTION	EXAMPLE
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	

#267 Explained Report Bookmark

Which operator does not work with float?

- A /
- B %
- C ++
- D !

Correct Answer :B

Explanation

Modulo Operator (%) in C programming Language.

As a result of using modulo operator on two operands, remainder will be the output. If floating type operands will be used for modulo operator, we will get floating point value as quotient and the remainder will be zero, which will not generate right answer. So if we use float operands in modulo operator, we always get answer 0. That is why % is not used with float data type values.

#268 Explained Report Bookmark

Arrange the following operators in the precedence order of highest to lowest:

(a) ==

(b) ++

(c) %

(d) &

The correct order of precedence is:

- **A**
(b), (c), (a), (d)
- **B**
(d), (b), (c), (a)
- **C**
(a), (d), (b), (c)
- **D**
(c), (a), (d), (b)

Correct Answer :A

Explanation

Concept:

When two operators share operands, then operator with higher precedence will be evaluate first.

Explanation:

“==” This is equality operator. It used in conditions to check the equality of two operands. It gives value true or false in case two operands are equal or not.

“++” This is increment operator. It always increments the value of operands by one. Example: if it is written as `a++`, then it means $a = a + 1$.

“%” This is modulus operator. It provides the remainder after division of two operands. Example: if $5 \% 4$ is written, then answer will be 1 (as remainder will be 1).

“&” it is bitwise AND operator. It does the AND of every bit of two operands. Result of AND operator is 1 only when both bits are 1.

Precedence of these operators are:

- `++` has higher precedence
- then modulus operator is evaluated
- then `==` precedence
- then `&` (bitwise AND) operator

So, correct option is (A): b, c, a, d

#269 [Explained](#) [Report](#) [Bookmark](#)

What is associativity for comma operator?

- A
Left to Left

- **B**
Left to Right
- **C**
Right to Left
- **D**
Right to Right

Correct Answer :B

Explanation

Comma operator (,) has associativity from left to right. It returns the rightmost operand in the expression and it simply evaluates the rest of the operands and finally rejects them.

#270 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
{  
    printf("%d", 5/0 );  
}
```

- **A**
5
- **B**
Compilation error
- **C**
32767
- **D**
Runtime error

Correct Answer :D

Explanation

_ Division by zero is an exceptional condition in C programming language and this will cause a runtime error.

#271Explained Report Bookmark

which statement is true regarding function?

- **A**
Any C program contains at least one function
- **B**
A function can be defined inside another function.
- **C**
In a C program there can be more than one functions with the same name
- **D**
A C program can be written without functions

Correct Answer :A

Explanation

Any C program always starts with main() function only. So it is necessary to have main() function in the program. This means that to execute any C program there should be at least one function which will be main() function only.

#272Explained Report Bookmark

The declaration “void fun(int);” indicates that the function

- A
returns a float value
- B
has no arguments
- C
returns nothing
- D
has default arguments

Correct Answer :C

Explanation

In computer programming, when void is used as a function return type, it indicates that the function does not return a value.

#273 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statement is true regarding recursive function?

- a) It is also called as a circular definition
 - b) It occurs when a function calls another function more than once
 - c) It occurs when a statement within function calls itself
 - d) A recursive function can not have a return statement within it
- A
a & c

- **B**
a & b
- **C**
b & d
- **D**
a , b, & d

Correct Answer :A

Explanation

A recursive function is a function that calls itself during its execution

#274[Explained](#) [Report](#) [Bookmark](#)

What is false about function main?

- **A**
main can have parameters
- **B**
main can return a value
- **C**
main can be called
- **D**
variables defined inside main become global variables

Correct Answer :D

Explanation

variables defined inside main are local variables to main and they can be accessed from inside of main only. They cannot be accessed from outside.

#275[Explained](#) [Report](#) [Bookmark](#)

A function has by default returning data type_____.

- **A**
void
- **B**
char
- **C**
int
- **D**
none of these

Correct Answer :C

Explanation

If we do not specify a return type, C will implicitly declare it as int.

#276 [Explained](#) [Report](#) [Bookmark](#)

What is true regarding a returning data type of a function?

- **A**
Function without returning data type can not return a value
- **B**
Function having returning data type void can return a value
- **C**
Function having returning data type void can not have return statement
- **D**
none of these

Correct Answer :D

Explanation

Void functions are created and used just like value-returning functions except they do not return a value after the function executes

Four Categorie

1 Function with no argument and no return value: Void Function

2 Function with arguments but no return value: Void Function

3 Function with no arguments but returns a value: Return Function that returns a value

4 Function with arguments and return value : Return Function that returns a value

#277 [Explained](#) [Report](#) [Bookmark](#)

What we can have in function name?

- A
blank
- B
underscore
- C
special char
- D
#

Correct Answer :B

Explanation

A function name is an identifier which cannot be formed with the help of special characters but can contain underscore in it.

#278 [Explained](#) [Report](#) [Bookmark](#)

The declaration “int *P(char a[]);” indicates

- A P is pointer to integer
- B P is pointer to function that returns an integer
- C P is function that returns an integer pointer
- D Illegal declaration of

Correct Answer :C

Explanation

int * P(char a[]) means P is a function that has an array as formal argument and this function will return a pointer which contains an address of an integer.

#279 [Explained](#) [Report](#) [Bookmark](#)

What happens in call by value?

- A Changes made in formal Parameters are reflected in actual parameters
- B Changes made in formal Parameters are not reflected back in actual parameters
- C Addresses of actual parameters are passed to formal parameters

- **D**
Values of actual parameters are passed to local variables of a called function

Correct Answer :B

Explanation

In the scenario of call by value, changes made in formal parameters are not reflected back in actual parameters and the formal parameters are local variables, which exist during the execution of the function only, and are only known by the called function

#280[Explained](#) [Report](#) [Bookmark](#)

How many times & when memory is allocated to formal parameters?

- **A**
Only once during compilation time
- **B**
Only once during execution time, at the start of program
- **C**
Only once during executing time, before function call
- **D**
For every call of that function

Correct Answer :D

Explanation

Memory is allocated to a function each time when it is called in the program.

#281[Explained](#) [Report](#) [Bookmark](#)

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

Correct Answer :D

Explanation

(A) With pointers, address of variables can be passed different functions can use this address to access the variables.

(B) When large structure variables passed or returned, they are copied as everything is passed and returned by value in C. This can be costly with structure containing large data. To avoid this copying of large variables, we generally use pointer for large structures so that only address is copied.

(C) With pointers, we can implement "linked" data structures. Java uses reference variables to implement these data structures. Note that C doesn't support reference variables.

#282 Explained Report Bookmark

Pick the correct statements.

I. **The body of a function should have only one return statement.**

II. The body of a function may have many return statements.

III. A function can return only one value to the calling environment.

IV. If return statement is omitted, then the function does its job but returns no value to the calling environment.

- **A**
I and II
- **B**
II and III
- **C**
I and III
- **D**
II and IV

Correct Answer :B

Explanation

IIInd statement can be proven with an example of if and else. If can return one value and else can return another. But which one will be passed to the calling environment depends on the if condition only i.e. if the if block return true then the return value mentioned inside the if block will be executed otherwise else block will be executed.

IIrd statement is true too. First encountered return statement will be executed only. Others will be ignored.

Example:

```
if(a>b) {
```

```
    return a;}
```

```
else {  
    return b;}
```

#283[Explained](#) [Report](#) [Bookmark](#)

For every function call in a program, compiler require_____.

- **A**
its definition
- **B**
its declaration
- **C**
both, its declaration & definition
- **D**
nothing

Correct Answer :B

Explanation

declaration are required to call a function in a program. Declaration helps to check order of arguments and what type of value will be returned

#284[Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void C( ) { printf("C"); }  
  
void B( ) { C( ); printf("B"); }  
  
void A( ) { printf("A"); B( ); }  
  
void main( ) { printf("M"); A( ); }
```

- A CAMB
- B MABC
- C MCAB
- D MACB

Correct Answer :D

Explanation

Program always starts from main() function. From main() function, A() function is called. From function A(), function B() is called and finally from function B(), function C() is called.

#285 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int transfer(int a, int b)

{

    char t=1;

    t=a; a=b; b=t;

    return a;
```

```

    return b;
}

void main( )

{
    int x=2,y=3;

    x=transfer(x,y);

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

```

- **A**
2 3
- **B**
3 2
- **C**
2 2
- **D**
3 3

Correct Answer :D

Explanation

In this program, transfer() function is called with arguments x and y. 2 and 3 have been passed to transfer() function. As there are two return statements in the function transfer(), when first return statement is executed it will take

control to the calling function and cannot go back from where it has come and another return statement cannot be executed. Finally rest of the statements of the calling function will be executed and as a result of this x will contain 6.

#286 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int f(int a) { a>=20 ? return(10) : return(20); }

void main( )

{

    int b=5;

    b=f (b+15);

    printf("%d", b);

}
```

- **A**
10
- **B**
20
- **C**
5
- **D**
Error

Correct Answer :D

Explanation

Compilation error will occur here. Error will be like this

[Error] expected expression before 'return'.

#287Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

display(int c) { return(c++) ; }

void main( )

{
    int k=45; float f;

    f=display(k);

    printf("%f", f);
}
```

- **A**
45
- **B**

45.000

- **C**
45.000000
- **D**
none

Correct Answer :C

Explanation

In this program, the function display() is called with argument k and 45 is passed to this function. return(c++) means return c before incrementing it and therefore 45 is returned back and collected in f which is of float data type.

#288Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int check(int x, int y)

{
    if (x==y)

    {
        y=x+1;  return (y);

    }

    else  {
```

```
    x=y+1; return(x);  
  
}  
  
}  
  
void main( )  
  
{  
  
    int a=3,b=5;  
  
    printf("%d",check(a,b));  
  
}
```

- A
5
- B
7
- C
6
- D
9

Correct Answer :C

Explanation

In this program, the function check() is called with arguments a and b and 3 and 4 have been passed respectively. 6 is returned from the function check() and printed on the screen

#289 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

void main( )

{
    int j=1;

    if(!j) printf("recursive");

    else

    {   j=0;

        printf("non-recursive");

        main();
    }
}
```

- **A**
recursive
- **B**
non-recursive
- **C**
non-recursive recursive
- **D**
non-recursive printed infinitely

Correct Answer :D

Explanation

: In this program we get Logical NOT (!) which use to reverse the operand and the non-zero value convert into zero

Then it go on else statement and print non-recursive

And after it call main

That's why it go in infinite loop, because j value become 0 everytime and it always execute the else statement

#290Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int sum(int a, int b)

{
    if(a==b)
        return (a+b);

    else
        return (a-b);
```

```
}

void main( )

{

    static int x;

    x=sum(x, 1);

    printf("%d", x);

}
```

- **A**
-1
- **B**
1
- **C**
2
- **D**
0

Correct Answer :A

Explanation

In this program, we get -1 as our output

As per the rule program execution starts with main and we get a static variable that is not initialized, then it will initialize with zero. And x will call sum function with two-parameter which have value of x and 1

In the function the if condition becomes false and it returns -1 as output.

#291 [Explained](#) [Report](#) [Bookmark](#)

Pointer can store_____.

- A data
- B any address
- C data or address
- D any valid address and NULL

Correct Answer :D

Explanation

In c pointer is used to store the address of a variable.

And we can assign null to a pointer both are valid in c

#292 [Explained](#) [Report](#) [Bookmark](#)

Q contains 100 and P contains 104,

if P-Q is 2 then what type of pointers are they?

- A char *
- B int *
- C float *

- D
void *

Correct Answer :B

Explanation

It is int* type pointer

#293 [Explained](#) [Report](#) [Bookmark](#)

Which arithmetic operator is allowed with pointer?

- A
Subtracting 2 pointers
- B
adding two pointers
- C
Adding constant to a pointer
- D
Subtracting constant from a pointer

Correct Answer :A

Explanation

In arithmetic pointer subtracting of 2 pointer is allowed. Pointer addition, multiplication and division are not allowed as these are not making any sense in pointer arithmetic

#294 [Explained](#) [Report](#) [Bookmark](#)

If P is int * and contains address 100, then after p++ what it will contain?

- A
100
- B
101
- C
102
- D
103

Correct Answer :C

Explanation

`p++` means `p=p+1`. Here `p` contains 100 and `p+1` will be $(100+1*2)$ which is 102 and now `p` will contain 102 inside itself.

#295 [Explained](#) [Report](#) [Bookmark](#)

By default pointer is_____.

- A
near
- B
far
- C
huge
- D
smart

Correct Answer :A

Explanation

All default pointers are called as near pointers (16 bit pointers) . the maximum address store in near pointer is 65565

Eg : int *ptr; and int near *ptr both are same

#296 [Explained](#) [Report](#) [Bookmark](#)

Size of far pointer is _____ bytes.

- **A**
1
- **B**
2
- **C**
4
- **D**
depends on its data type

Correct Answer :C

Explanation

far pointer can point or access the whole memory of RAM means far pointer can access all 16 segments. And it's size is 4 bytes

#297 [Explained](#) [Report](#) [Bookmark](#)

Near pointer stores_____ address.

- **A**
only offset
- **B**
only segment
- **C**
segment and offset

- **D**
segment or offset

Correct Answer :B

Explanation

the near pointer points only to the 64 kb data segment .The size of the near pointer is two byte. And with the help of near we can create near pointer

Eg int near *ptr

#298 [Explained](#) [Report](#) [Bookmark](#)

Size of a segment can be maximum_____.

- **A**
64 bytes
- **B**
64 KB
- **C**
16 bits
- **D**
16 bytes

Correct Answer :B

Explanation

The max size is 64K. There is no minimum - you can use only first byte, but the segment is always 64K. Maybe the exception are the last segments - the latest (0xFFFF) is 16 bytes, before the latest are 32, 48, 64 bytes... Generally, 8086 segments are method for addressing of more than 64K with 16 bit address registers.

#299 Explained Report Bookmark

If address bus is 16 bit, How many memory locations can we address?

- **A**
16
- **B**
65535
- **C**
64
- **D**
65536

Correct Answer :D

Explanation

In many older computers, the address bus was 16 bits wide ($a = 16$). This meant that there were 16 wires. Such microprocessors could address up to $2^{16} = 65536$ memory locations. By increasing the width of the address bus, more memory locations can be directly addressed.

#300 Explained Report Bookmark

Generic pointer is _____.

- **A**
char
- **B**
generic*
- **C**
void *
- **D**
there is no such concept

Correct Answer :C

Explanation

void pointer is also known as generic pointer. Meaning of generic pointer is a pointer which can point to a type of data.

#301 [Explained](#) [Report](#) [Bookmark](#)

What is true about void *

- **A**
It can be referenced
- **B**
It can be dereferenced
- **C**
It can be incremented
- **D**
All are true

Correct Answer :A

Explanation

void* can be referenced but it is not deference in programming language

Because address of any variable of any datatype can be assigned to a void pointer variable

#302 [Explained](#) [Report](#) [Bookmark](#)

Only _____ pointer can store any type of address without typecasting.

- **A**
void
- **B**
char

- **C**
int
- **D**
this

Correct Answer :A

Explanation

void pointer is also called a generic pointer which means it does not have any data type [void means nothing]. It can store any address of any type.

#303 **Explained Report Bookmark**

Consider declaration void **P; What will happen for statement P++; if P contains address 100

- **A**
It will generate error for statement P++;
- **B**
Address will remain 100
- **C**
No error and P will become 102
- **D**
P will become 104

Correct Answer :D

Explanation

: pointer variables always store the address of another variable which is int type and the size depends on the compiler . according to 32 bit it take 4 byte

#304 Explained Report Bookmark

What is the size of char **p ?

- A 0 bytes
- B 1 byte
- C 4 bytes
- D 2 bytes

Correct Answer :C

Explanation

According to 32 bit it gives 4 bytes to the “ pointer to a pointer of type char. Char store 1 byte but here we use pointer that's why it is given 4 .

#305 Explained Report Bookmark

A[i] is same

- A $*(A + i)$
- B $*(i + A)$
- C $i[A]$
- D all of them (1, 2, 3)

Correct Answer :D

Explanation

$*(A+i)$ it means value at i

$*(i+A)$ also gives the same meaning so all of them are the same.

#306 [Explained](#) [Report](#) [Bookmark](#)

Which operator is used to access structure member through pointer?

- A
 *
- B
 .
- C
 ->
- D
 =>

Correct Answer :C

Explanation

:To access members of a structure using pointer , we use the $->$ operator.

#307 [Explained](#) [Report](#) [Bookmark](#)

Consider declaration int *P; Which of the following statement will generate Error?

- A
 $*++P$
- B
 $++*P$

- **C**
 $*P++$
- **D**
 P^{*++}

Correct Answer :D

Explanation

In this program $*{+}p$, ${+}*p$ and $*p{+}$ will work but $p^{*}{+}$ is not the write declaration

Option 1,2 and 3 are pointer expression

#308 [Explained](#) [Report](#) [Bookmark](#)

What is indicated by following declaration?

`char **P(int *A[]);`

- **A**
P is a Pointer to a function that return char *
- **B**
P is a Pointer to a function that has array of integer pointers as parameter
- **C**
P is a function that has array of integer pointers as parameter and returns char **
- **D**
P is a Pointer to Pointer to char typecasted in to array of int *

Correct Answer :C

Explanation

P is a function which will return a pointer which points to a pointer that has address of char data type.

#309 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
{  
    char *ch="sun";  
  
    printf("%c,%s", *++ch, ch);  
}
```

- A
s,sun
- B
t,sun
- C
u,sun
- D
u,un

Correct Answer :C

Explanation

The statement `char *ch="sun"` creates a string for the literal and then stores its address in the pointer variable `ch`. The pointer `str` now points to the first character of the string “sun”

`*++ch` can be written as `*(++ch)` because `*` and `++` both have same priority but associativity is from right to left. This will make the pointer to point next element of the string.

#310 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
{  
    char far *cash,*cheque;  
  
    printf("%d %d", sizeof(cash), sizeof(cheque));  
}
```

- A
2 2
- B
2 4
- C
4 2
- D
4 4

Correct Answer :C

Explanation

cah is far pointer which size is 4 byte.

By default chequeis near pointer which size is 2 byte.

#311 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    int a[ ]={1,2,3,4,5},j;  
  
    for(j=0; j<5; j++)  
  
    {  
  
        printf("%d", *(a+j)); a++;  
  
    }  
  
}
```

- **A**
1 2 3 4 5
- **B**
1 2 3 4
- **C**
1 2 3
- **D**
Compilation error

Correct Answer :D

Explanation

here we get compilation error because first in program It is not declare and we use extra semicolon

We can use exactly 2 semicolons in for loop.

#312 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    int count=10,*temp,sum=0;  
  
    temp=&count;  
  
    *temp=20;  
  
    temp=&sum;  
  
    *temp=count;  
  
    printf("%d %d %d", count, *temp, sum);  
  
}
```

- **A**
20 20 20
- **B**
20 20 0

- **C**
1020 0
- **D**
compilation error

Correct Answer :D

Explanation

error: `amp' undeclared (first use in this function)

#313Explained Report Bookmark

What will be the output of the following C code?

```
#include<stdio.h>

main()
{
    int a[3]
    [4]={1,2,3,4,4,3,2,1,1,3,4,1};
    printf("%d",*(a+1)+2);
}
```

- **A**
2
- **B**
1
- **C**
3

- D
4

Correct Answer :A

Explanation

multidimensional array (a+1) :- increments the value of array pointer by 1 that in turn points to row 2 of array(property of multidimensional array)

pointer as it points to array of pointers(which are pointing to 1D arrays).
(*a+1)+2 now points to exact same location as a[1][2].

#314 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    float floats[ ]={ 13,24,1,5,1,5,5,4,3,5}, *w, *boat;  
  
    w=floats;  
  
    boat=floats+4;  
  
    w=w*2;  
  
    boat=boat/2;  
  
    printf("%f %f", *w, *boat);
```

}

- A
13.000000 1.500000
- B
13.240000 1.500000
- C
13.000000 1.000000
- D
Error

Correct Answer :D

Explanation

Compilation error will be like this

[Error] invalid operands to binary * (have 'float *' and 'int')

[Error] invalid operands to binary / (have 'float *' and 'int')

#315 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    static char str[ ]="DAC Entrance" ;  
  
    char *s;
```

```
s=&str[5]-1;  
  
while(*s) printf("%c", *s++);  
  
}
```

- **A**
DAC Entrance
- **B**
Entrance
- **C**
ntrance
- **D**
Error

Correct Answer :B

Explanation

str[5]-1 means &str[4] i.e. address of element E which is stored at s. In the expression *s++, value at address s is printed first then increment of address to point next element is done later.

#316 Not Explained Report Bookmark

What will be the output of the following C code?

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

```
change(int *b)
```

```
{
```

```

int i;

for(i=0;i>4;i++)

{

    *b = *b+1; b++;

}

void main( )

{

    int a[5]={2,3,4,5,6},i;

    change(a);

    for (i=4;i>=0;i--)

        printf("%d",a[i]);

}

```

- **A**
5 4 3 2 1
- **B**
6 5 3 4 2
- **C**
6 5 4 3 2 1

- **D**
6 5 4 3 2

Correct Answer :D

No Explanation Available

#317 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

void main( )
{
    static int A[ ]={0, 1, 2, 3, 4};

    static int *P[ ]={A, A+1, A+2, A+3, A+4};

    printf("%d %d %d", **P, *(*(P+1)), *(*(P+0)));
}
```

- **A**
0 1 2
- **B**
0 1 0

- **C**
1 2 3
- **D**
Compilation error

Correct Answer :B

Explanation

$**p = *(A) = 0$

$*(*(p+1)) = *(p[1]) = *(A+1) = *(A[1]) = 1$

$*(*(p+0)) = *(p[0]) = *(A+0) = *(A[0]) = 0$

#318 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

void main( )
{
    void showval(int, float);

    void(*s1)(int, float);

    s1=showval;
```

```
(*s1) (10, 3.5);  
}  
  
void showval(int i, float f) { printf("%d %f", i, f); }
```

- A
10 3.0
- B
10 3.5
- C
10.0
- D
Syntax error

Correct Answer :B

Explanation

```
void(*s1)(int, float);
```

Create a pointer to the method with int and float arguments in it.

```
s1=showval;
```

Share values of s1 with showval method

```
(*s1)(10, 3.5);
```

Pass 10 and 3.5 as an integer and float respectively to the method which is pointed by s1 pointer.

```
void showval(int i, float f) { printf("%d %f", i, f); }
```

Received 10 and 3.5 will be printed here.

#319 [Not Explained](#) [Report](#) [Bookmark](#)

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

```
#include <stdio.h>

void main( )

{
    int showfunc( );
    int (*func) ( );
    func=showfunc;
    display(func);
}

showfunc( ) {printf("CCAT ..."); }

display(f1)
int (*f1)( );
{
    (*f1)( );
}
```

}

- **A**
CCAT ... is printed once
- **B**
CCAT ... is printed twice
- **C**
no output
- **D**
error

Correct Answer :A

No Explanation Available

#320 [Explained](#) [Report](#) [Bookmark](#)

In worst case, how many times will do – while loop executes (minimum)?

- **A**
0
- **B**
1
- **C**
n times
- **D**
infinite

Correct Answer :B

Explanation

In C and C++ a “`do .. while`” loop has the condition at the end. It is tested first after executing the body once. Consequently if the condition is `false` the body will execute exactly once.

#321 Explained Report Bookmark

In worst case, how many times while loop executes (minimum)?

- A
0
- B
1
- C
N TIMES
- D
infinite

Correct Answer :A

Explanation

Depends on how you write it. If while(){} , then yes, the minimum number of times is 0. If not talking about the DO, then it's 0. Yes, if the while's condition isn't satisfied at the first time, the loop is executed zero times

#322 Explained Report Bookmark

Which loops are similar in behavior?

- A
for & do while
- B
while & do while
- C
while & for
- D
do while, for & while

Correct Answer :C

Explanation

Loops are a way to repeat the *same* code multiple times. ... the *for loop* to *while* without altering its *behavior* (the output should stay *same*).

#323 [Explained](#) [Report](#) [Bookmark](#)

Which loop allows to omit condition?

- **A**
while
- **B**
do while
- **C**
for
- **D**
none

Correct Answer :C

Explanation

The for loop specification then goes on to allow you to omit clause-1 so that you can declare or initialize variables elsewhere, and you can omit expression-3 so that you are not required to evaluate any expression upon completion of each loop

#324 [Explained](#) [Report](#) [Bookmark](#)

Which loop does not require to write ';' after loop condition?

- **A**
while
- **B**
do while

- **C**
for
- **D**
none

Correct Answer :A

Explanation

While the `condition` is truthy, the `code` from the loop body is executed.

#325 [Explained](#) [Report](#) [Bookmark](#)

Missing condition in for loop will generate_____.

- **A**
Compiler error
- **B**
Runtime error
- **C**
Warning
- **D**
Infinite loop

Correct Answer :D

Explanation

An expression to be evaluated before each loop iteration. If this expression evaluates to `true`, statement is executed. This conditional test is optional. If omitted, the condition always evaluates to `true`

#326 [Explained](#) [Report](#) [Bookmark](#)

Missing condition in while loop will generate_____.

- **A**
Compiler error
- **B**
Runtime error
- **C**
Warning
- **D**
Infinite loop

Correct Answer :A

Explanation

Empty while loop is not allowed it will throw an exception

#327 [Explained](#) [Report](#) [Bookmark](#)

Which statement can stop the loop?

- **A**
continue
- **B**
break
- **C**
initialization
- **D**
if

Correct Answer :B

Explanation

The break in C or C++ is a loop control statement which is used to terminate the loop.

#328 [Explained](#) [Report](#) [Bookmark](#)

What continue does?

- A It helps to continue execution of a program
- B It skips that iteration & continues loop
- C It continues that iteration & then takes control outside loop
- D It continues execution of a function by skipping a loop

Correct Answer :B

Explanation

The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately.

#329 [Explained](#) [Report](#) [Bookmark](#)

What is correct execution sequence in a for loop?

- A loop statements, loop expression, loop condition, initialization
- B loop expression, loop statements, loop condition
- C loop statements, loop condition, loop expression
- D loop condition, loop statements, loop expression

Correct Answer :D

Explanation

The syntax of a for loop in C programming language is – for (init; condition; increment) { statement(s); } Here is the flow of control in a 'for' loop – The init step is executed first, and only once

#330 [Explained](#) [Report](#) [Bookmark](#)

What is not a difference between while & do-while loop?

- **A**
In while loop condition is checked 1 st & in do while it is checked last
- **B**
do-while loop executes at least once whereas while loop may not execute in worst case
- **C**
while loop stops when condition is false whereas do-while stops when condition is true
- **D**
In while loop, condition decides execution of current iteration whereas in do while, it decides execution of next iteration.

Correct Answer :C

Explanation

If the condition in a while loop is false, not a single statement inside the loop is executed. In contrast, if the condition in 'do-while' loop is false, then

also the body of the loop is executed at least once then the condition is tested

#331 [Explained](#) [Report](#) [Bookmark](#)

In for(...) how many semicolons are allowed?

- A
 ≤ 2
- B
exactly 2
- C
 ≥ 2
- D
 ≥ 3

Correct Answer :B

Explanation

There are two semicolons required. One after initialization and other after condition.

#332 [Explained](#) [Report](#) [Bookmark](#)

In do while loop, 5 th time condition is checked & now resulted into false, so how many times loop has executed?

- A
4
- B
5
- C
6
- D
0

Correct Answer :B

Explanation

In do-while loop, one time loop is executed before checking the condition and therefore in the given situation 5 times loop will be executed.

#333 [Explained](#) [Report](#) [Bookmark](#)

In while loop, 7 th time condition is checked & now resulted into false, so how many iterations executed?

- A 0
- B 5
- C 6
- D 7

Correct Answer :C

Explanation

In Loop, the statement needs to be written only once and the loop will be executed 6 times (7 one consider as false)as shown below

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

```
void main( )
```

```
{  
  
    int i = 1;  
  
    while (i < 7)  
  
    {  
  
        printf( "i= %d\n", i);  
  
        i++;  
  
    }  
  
}
```

#334 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

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

```
void main( )
```

```
{  
  
    int k=3;
```

```

while(k)

{

    int k=1;

    printf("%d ", k);

    k--;

}

k--;
}

```

- **A**
1
- **B**
3 2 1
- **C**
Infinite loop
- **D**
Error

Correct Answer :C

Explanation

Here there are two k variables. One k's scope is whole main function but another k's scope is inside only a block in which it is declared and defined. k variable inside a block has higher precedence than outer k inside that block and due to this only k variable of block comes under execution again

and again and causes infinite loop. There is nothing to do with outer k i.e. no manipulation is done on outer k and it remains 3 only.

#335 [Explained](#) [Report](#) [Bookmark](#)

Which operator works like if else?

- A ::
- B ?:
- C ->
- D >>

Correct Answer :B

Explanation

The ?: is a conditional operator. It is a ternary operator and requires three operands. It behaves like if-else. This operator works such that if given expression which is written before ? is true then the first condition is executed otherwise second condition is executed.

#336 [Explained](#) [Report](#) [Bookmark](#)

Which operator cannot be used for joining two conditions?

- A &&
- B ||
- C !

- **D**
both A and B

Correct Answer :C

Explanation

The ! is a logical NOT operator. ! operator is used to complement the condition under consideration. ! returns TRUE when condition is FALSE and returns FALSE when condition is TRUE.

#337 [Explained](#) [Report](#) [Bookmark](#)

Which is correct statement?

- **A**
alone if can occur without else
- **B**
alone else can occur without if
- **C**
else should be written before if
- **D**
else is compulsory for every if

Correct Answer :A

Explanation

The else is an optional part. During the program execution, the statements written inside the if block will only be executed when the condition mentioned is true. Otherwise, if the condition is false, the next consecutive lines after the if block will be executed. It will not search for else or generate an error.

#338 Explained Report Bookmark

What is true about nested if-else?

- **A**
we can write if-else only inside if part
- **B**
we can write if-else only inside else part
- **C**
we can write if-else in both if & else part
- **D**
we can write one if-else after another if-else

Correct Answer :C

Explanation

In nested if-else, inner if-else can be present in if part or else part or both of outer if-else.

#339 Not Explained Report Bookmark

Which nested if-else can be converted to switch?

null

- **A**
every nested if-else can be converted in to switch
- **B**
having 1 operand common in all conditions & other operand uncommon(contant)
- **C**
having relational operator == in all conditions
- **D**
both 2 & 3

Correct Answer :D

No Explanation Available

#340 [Explained](#) [Report](#) [Bookmark](#)

What is possible?

- **A**
writing switch in if-else
- **B**
writing if-else in switch
- **C**
writing one switch another switch
- **D**
all of the above are possible

Correct Answer :D

Explanation

Inside if-else, switch can be placed and if-else can be placed inside a switch as well. It is permitted to nest switch statements.

#341 [Explained](#) [Report](#) [Bookmark](#)

What is not required for writing switch?

- **A**
default
- **B**
case
- **C**
break

- **D**
continue

Correct Answer :D

Explanation

Switch has flow such that if any case matches, then this case's block come into flow and if the break statement is not there to indicate the end of this case, every case below this case would come into flow, until another break. So, as you can see there seems to be no use of continue here.

#342 [Explained](#) [Report](#) [Bookmark](#)

What is true about default?

- **A**
default can be written anywhere in switch
- **B**
default must be last case in switch
- **C**
default must be first case in switch
- **D**
default is compulsory in switch

Correct Answer :A

Explanation

The default statement is optional, and can appear anywhere inside the switch block. In case, if it is not at the end, then a break statement must be kept after the default statement to omit the execution of next case statement.

#343 Explained Report Bookmark

What is most false about switch?

- **A**
All cases require continue
- **B**
All cases require break
- **C**
Last case doesn't require break
- **D**
First case may not have break

Correct Answer :A

Explanation

Switch is not considered as loop so you cannot use continue inside a case statement in switch.

#344 Explained Report Bookmark

Switch can not handle _____ data type

- **A**
char
- **B**
int
- **C**
float
- **D**
float & double

Correct Answer :D

Explanation

The expression used in switch must be integral type (int, char and enum).

Any other type of expression is not allowed.

#345 [Explained](#) [Report](#) [Bookmark](#)

What is false about switch cases?

- A cases have to be constant expression
- B cases can be fractional
- C cases can be strings
- D both B & C

Correct Answer :D

Explanation

The expression used in switch must be integral type (int, char and enum).

Any other type of expression is not allowed.

#346 [Explained](#) [Report](#) [Bookmark](#)

What is true about switch cases?

- **A**
more than 1 cases having same action can be written together separated by comma
- **B**
more than 1 cases can be joined by using & &;, ||
- **C**
duplicate cases are not allowed
- **D**
small case & upper case are treated as same case

Correct Answer :C

Explanation

A switch statement tests the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed. Each case in a block of a switch has a different name/number which is referred to as an identifie

#347 [Explained](#) [Report](#) [Bookmark](#)

switch is preferred over nested if-else because

- **A**
It increases complexity of a program
- **B**
It improves execution speed of a program
- **C**
It increases readability of a program
- **D**
It reduces size of compiled code

Correct Answer :C

Explanation

A switch statement is usually more efficient than a set of nested ifs. ... The compiler can do this because it knows that the case constants are all the same type and simply must be compared for equality with the switch expression, while in case of if expressions, the compiler has no such knowledge.

#348 [Explained](#) [Report](#) [Bookmark](#)

Which character is used in switch to separate case from its action part

- A blank
- B colon
- C semi-colon
- D comma

Correct Answer :B

Explanation

No Explanation is required here, its a straight forward question

#349 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

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

```
void main( )
```

```

{
    int m=10, n=15;

    if( ! ( !m) &&m) printf("%d", m++);

    else printf("%d", n++);
}

```

- **A**
15
- **B**
10
- **C**
12
- **D**
16

Correct Answer :B

Explanation

```
*****
*****
```

! is a type of Logical Operator and is read as “NOT” or “Logical NOT”.

This operator is used to perform “logical NOT” operation,

i.e. the function similar to Inverter gate in digital electronics.

Syntax: ! Condition // returns true if the conditions is false
// else returns false

```
*****  
***** /
```

```
#include <stdio.h>

void main( )
{
    int m=10, n=15;
    printf("m=%d, !m);
    if(!(!m) && m)
        // (!(!10) && 10)
        // = > !(0) && 10 -- !(0) = 1
        // = > 1 && 10
        // so if i evaluated as true
```

```
    printf("m=%d", m++) ; // first 10 will print then  
incremented  
  
else  
  
printf("n=%d", n++) ;  
  
}
```

#350 Explained Report Bookmark

What is the output of the below code snippet?

```
#include <stdio.h>  
  
void main( )  
  
{  
  
int m=0, n=-1;  
  
if(~m && ~n) printf("%d", n);  
  
else printf("+%d", ~m);  
  
}
```

- A
+1
- B
+1

- **C**
-1
- **D**
+32767

Correct Answer :A

Explanation

The `~` is a Bitwise NOT operator. Bitwise NOT is a unary operator. Its job is to complement each bit one by one. Result of NOT is 0 when bit is 1 and 1 when bit is 0. Table of Bitwise NOT is shown below:

A

`~A`

0

1

1

0

#351 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

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

```

void main( )

{
    int k=4;

    switch(k)

    {
        default: printf(" A ");
        case 1: printf(" B ");
        case 4: printf(" C ");
        case 2+3-1: printf(" D ");
    }
}

```

- **A**
C
- **B**
D
- **C**
C D
- **D**
Error

Correct Answer :D

Explanation

error: duplicate case value is not allowed in switch statements

#352 Explained Report Bookmark

Which storage class is used to increase scope of a variable?

- A auto
- B static
- C extern
- D register

Correct Answer :C

Explanation

Extern storage class is used to extend the visibility of variables where variable is defined somewhere else (in same file or different file) in the program.

#353 Explained Report Bookmark

Which function allows format specifiers?

- A clrscr
- B strlwr
- C strchr
- D scanf

Correct Answer :D

Explanation

The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf(). Some examples are %c, %d, %f, etc.

#354 [Explained](#) [Report](#) [Bookmark](#)

By default int is _____ & _____.

- A short & unsigned
- B long & unsigned
- C short & signed
- D long & signed

Correct Answer :C

Explanation

short: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).

int: By default, the int data type is a 32-bit signed two's complement integer, which has a minimum value of -2³¹ and a maximum value of 2³¹-1

#355 [Explained](#) [Report](#) [Bookmark](#)

How many ASCII characters are there?

- **A**
256
- **B**
128
- **C**
255
- **D**
127

Correct Answer :C

Explanation

total number of Character in ASCII table is 256 (0 to 255). 0 to 31(total 32 character) is called as ASCII control characters (character code 0-31). 32 to 127 character is called as ASCII printable characters (character code 32-127). 128 to 255 is called as The extended ASCII codes (character code 128-255).

#356 [Explained](#) [Report](#) [Bookmark](#)

Maximum integer range is_____.

- **A**
32768
- **B**
32767
- **C**
65536
- **D**
255

Correct Answer :B

Explanation

It is the highest number that can be represented in a signed 16-bit integer (to be exact, the range is $2^{15}-1$ to -2^{15} or 32767 to -32768),

#357 [Explained](#) [Report](#) [Bookmark](#)

What is used to access individual element of an array?

- **A**
()
- **B**
[]
- **C**
{ }
- **D**
->

Correct Answer :B

Explanation

Array elements are accessed by using an integer index with Square bracket []. Array index starts with 0 and goes till size of array minus 1.

#358 [Explained](#) [Report](#) [Bookmark](#)

Which operator does not check 2nd condition if 1st condition is false?

null

- **A**
||

- **B**
|
- **C**
&
- **D**
&&

Correct Answer :D

Explanation

&& returns TRUE or 1 when all the conditions under consideration are true and returns FALSE or 0 when any one or more than one condition is false. && does not check 2nd condition if 1st condition is false due to short circuit evaluation. Short circuit evaluation in case of && simply means if there is a condition anywhere in the expression that returns false, then the rest of the conditions after that will not be evaluated.

#359 [Explained](#) [Report](#) [Bookmark](#)

Which is not an operator in C?

null

- **A**
->
- **B**
*
- **C**
!=
- **D**
+=

Correct Answer :B

Explanation

The `->` is called the arrow operator. This operator is used in C to access members of a structure which is pointed by the pointer. The not-equal-to operator (`!=`) in C returns true if the operands do not have the same value; otherwise, it returns false. The `+=` operator in C first adds the current value of the variable on left to the value on right and then assigns the result to the variable on the left. Therefore `.*` is not a valid operator in C programming language.

#360Explained Report Bookmark

What will be the output of the following C code?

```
void main( )  
  
{  
  
    float a=5,b=2;  
  
    printf("%f", a%b*2);  
  
}
```

- **A**
2.0
- **B**
Compilation error
- **C**
1.0
- **D**
0.5

Correct Answer :B

Explanation

Compilation error will be like this [Error] invalid operands to binary % (have 'float' and 'float'). This is because modulo operator (%) is a binary operator and applicable on integer data type operands/values but in the above question float data type operands/values are given.

#361 [Explained](#) [Report](#) [Bookmark](#)

Which statement will generate an error?

- **A**
a=b>c ? b : c;
- **B**
(b>c) ? a=b : a=c;
- **C**
(b>c) ? a=b : (a=c);
- **D**
(b>c) ? (a=b) : (a=c);

Correct Answer :B

Explanation

lvalue is required here. Compilation error will be like this [Error] lvalue required as left operand of assignment.

#362 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )
```

```
{  
  
    int no_1 =0;  
  
    printf("%d", 5/no_1);  
  
}
```

- **A**
5
- **B**
Compiaction error
- **C**
32767
- **D**
Rnntime error

Correct Answer :B

Explanation

There are multiple errors occur because of both of the statements given in the program. Compilation errors will be

[Error] expected '=', ',', ';', 'asm' or '__attribute__' before numeric constant

[Error] lvalue required as left operand of assignment

[Error] 'no' undeclared (first use in this function)

[Note] each undeclared identifier is reported only once for each function it appears in

Error] expected ')' before numeric constant

All these errors occur due to wrong or illegal variable name is used during declaration of a variable.

#363 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    int x=1, y=2, z=6;  
  
    x+=x+y*z=8;  
  
    printf(" %d", x);  
  
}
```

- A
18
- B
2
- C
14
- D
Compilation error

Correct Answer :D

Explanation

Compilation error will be [Error] lvalue required as left operand of assignment.

lvalue simply means an object that has an identifiable location in memory(i.e. having an address). In any assignment statement, “lvalue” must have the capability to hold the data. lvalue cannot be a function, expression or a constant.

#364 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    int x=2;  
  
    float f1=1.0, f2=2.0, sum;  
  
    sum=x*f1+f2/x+f1*f2;  
  
    printf("%f ", sum);  
  
}
```

- A
5.000000

- **B**
6.000000
- **C**
1.500000
- **D**
Compilation error

Correct Answer :A

Explanation

Given expression can be written as $(x*f1) + (f2/x) + (f1*f2)$ due to the highest priority of multiplication or division operator over addition operator. First of all, all the brackets are evaluated from left to right and then addition will be done from left to right. As sum is of float data type, resultant value will be assigned as float due to %f to sum variable.

#365Explained Report Bookmark

What will be the output of the following C code?

```
void main( )  
{  
    int v=-5, w=0, x=4;  
  
    printf("%d", w || v || x&&w);  
}
```

- A
1
- B
0
- C
5
- D
Garbage value is printed

Correct Answer :A

Explanation

The given expression can be written as $w \mid\mid v \mid\mid (x \&\& w)$. First of all, bracket will be evaluated because priority of $\&\&$ is higher than $\mid\mid$. After evaluation of bracket, expression will be $w \mid\mid v \mid\mid 0$. Now the given expression is checked from left to right. Now w has 0 which means false and then due to this v has to be checked and v results as true because it contains non-zero value. So the whole expression results as true which means 1 as the output.

#366 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
{  
    int s1 = 1, s2=4;  
  
    printf("%d", (s1&s2)?5:10 );  
}
```

- A
10
- B
5
- C
0
- D
1

Correct Answer :A

Explanation

In the given expression, `(s1&s2)` is evaluated first. `&` is a bitwise And operator in C. The output of `(s1&s2)` is 0 which means false in C. Due to the false of the condition, second operand will be evaluated and 10 will be produced as the output of the whole expression.

#367 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
{  
    int i=2, j=3, k=0, m=0 ;  
  
    float a=1, b=2;  
  
    k=i/j *k;
```

```
m=j/i*j;  
  
a=i*/j*j;  
  
b=i*j/k;  
  
printf("%d %d %f %f ", k, m, a, b);  
  
}
```

- **A**
0 3 0.000000 0.000000
- **B**
2 3 0.000000 2.000000
- **C**
0 3 0.000000 2.000000
- **D**
Error

Correct Answer :D

Explanation

error will occur due to the expression i^*/j^* . Error will be like this [Error] expected expression before '/' token.

#368 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )
```

```
{  
  
    int xx=3*(5 +1) / (2-2*10);  
  
    printf("%d", xx);  
  
}
```

- A
1
- B
-1
- C
0
- D
Divide by zero error

Correct Answer :B

Explanation

In the given expression, brackets will be evaluated first. $(5+1)$ gives 6. $(2-2*10)$ gives -18. Now the expression becomes $3*6/-18$. Finally this expression gives -1 as the output.

#369 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )
```

```
{
```

```
int x=32767, y=9;  
  
x=x+y;  
  
printf("%d", x);  
  
}
```

- **A**
32776
- **B**
-32776
- **C**
-32760
- **D**
Garbage value

Correct Answer :A

Explanation

In the given expression $x+y$ will be evaluated first and the output will be assigned to x .

#370 [Explained](#) [Report](#) [Bookmark](#)

[Find invalid bitwise operator](#)

- A
^
- B
>>
- C
~
- D
&&

Correct Answer :D

Explanation

&&, because we have 6 bitwise operators that are :

- ^ (Bitwise XOR)
- &(Bitwise AND)
- | (Bitwise OR)
- ~(Bitwise NOT)
- <<(Bitwise left shift)
- >>(Bitwise Right shift)

#371 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
. void main( )
{
    int ch=26;
    printf("%X", ch );
}
```

```
}
```

- **A**
26
- **B**
1a
- **C**
1A
- **D**
Error

Correct Answer :C

Explanation

: Here answer is 1A, because we use %X in format string and %X is used for hexadecimal format and here we use capital X which is used to print the value to uppercase hexadecimal format.

#372 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
{  
    int d=3.142;  
  
    printf("%d", 'd' );  
  
}
```

- A
100
- B
142000
- C
3
- D
4

Correct Answer :A

Explanation

: output is 100 , because in the argument list we use character and we write character in single quotation. So it will print the ASCII value of d which is 100.

#373 [Explained](#) [Report](#) [Bookmark](#)

How to store the value of x in y?

- A
`x=y;`
- B
`) x= =y;`
- C
`y=x;`
- D
`y= =x;`

Correct Answer :C

Explanation

when we assign the value of x in y , then x will come at the right hand side . Then it will store the value of x in variable y.

#374 [Explained](#) [Report](#) [Bookmark](#)

The maximum length of an identifier is_____.

null

- A 10 char
- B 32 char
- C 8 char
- D Compiler dependant

Correct Answer :B

Explanation

Identifier is a name of variable or constant which is not more than 32 char.

#375 [Explained](#) [Report](#) [Bookmark](#)

Octal constants are preceded by char_____.

null

- A 0(zero)

- **B**
0
- **C**
0x
- **D**
ox

Correct Answer :A

Explanation

In c language octal are preceded by zero(0) and other 0X and 0x are used for hexadecimal numbers.

And o is the format specifier to display the octal number so ans is option 1) zero (0)

#376 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )
{
    printf("%d", sizeof(char));
    printf("%d", sizeof(int));
    printf("%d", sizeof(float));
}
```

- A
1 4 4
- B
2 4 4
- C
1 2 4
- D
Error : invalid parameter to sizeof

Correct Answer :A

Explanation

sizeof() operator is used to get the size of variable .In C the size of char is 1 byte, int = 4 byte and float = 4 byte

#377 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
int x = 10;

void main( )
{
    printf("%d", x);

    int x = 20;
}
```

```

        int x =30;

        printf("%d",x);

    }

printf("%d",x);

}

```

- **A**
10 20 30
- **B**
10 30 20
- **C**
10 30 30
- **D**
Error

Correct Answer :B

Explanation

In first printf it will take “10” as the value of x , then in next line value of x updated and it become 20 , again it update in next line and now x is 30 but this 30 will only access in the innermost curly bracket so next it will print “30” and after that we come out of the innermost block so the value of x is 20 , then the last printf will print “20”.

#378 Explained Report Bookmark

What will be the output of the following C code?

```
void main( )  
  
{  
  
    printf("%d", printf("*sun*") );  
  
}
```

- A
 *sun*5
- B
 sun
- C
 5*sun*
- D
 error

Correct Answer :A

Explanation

In this program first printf will execute and it print *sun* and in next printf we use %d to read a integer value then printf return the number which is successfully print on the console and if we count *sun* then it is 5 and that will give us final output as -- *sun*5

#379 Explained Report Bookmark

What will be the output of the following C code?

```
void main( )  
  
{  
  
    printf("\\\\65");  
  
}
```

- A
\\65
- B
\\A
- C
\\\\65
- D
\\5

Correct Answer :D

Explanation

In c language \\ (double slash) is used to print single slash on console so starting two slash will use to print \\ as output and then we get \\6 which will not print on console .so our final output is \\5

#380Explained Report Bookmark

What will be the output of the following C code?

```
void main( )  
  
{
```

```
    printf("%d %d %d", 8 );  
}  
  
}
```

- **A**
8 garbage garbage
- **B**
8 8 garbage
- **C**
8 0 garbage
- **D**
error

Correct Answer :A

Explanation

when we don't give value in the argument list then it will take garbage value . so 8 will go with first %d and then we get two garbage values

#381 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include<stdio.h>  
  
void main() {  
  
    int a = 9;  
  
    if (a = 5) {
```

```
    printf("It is important to be nice");

} else {

    printf("It is nice to be important");

}

}
```

- **A**
It is important to be nice
- **B**
It is nice to be important
- **C**
None of the above
- **D**
Error

Correct Answer :A

Explanation

In this program first we assign the value of x is 9 then in if condition we reassign the value and it is non-zero value so the condition will become true and it will print the if statement that is – It is important to be nice

#382 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
void main() {  
  
    int i;  
  
    for (i = 1; i < 10; i++) {  
  
        if (i == 3)  
  
            continue;  
  
        printf("%d", i);  
  
    }  
  
}
```

- **A**
12456789
- **B**
2345
- **C**
123456789
- **D**
123456910

Correct Answer :A

Explanation

In this program, we will start our loop with 1 and then check whether it is less than 10 or not. when we get i less than 10 it enters in the loop and checks condition whether the value of I equal to 3 or not

If it is not equal to 3 it will not execute the next line [In the if-else condition when there is only one line statement then it is not mandatory to use curly bracket] and print the current value of i

After printing it will Increment the value of i by 1 then same process will get execute and when the inner condition become true it will continue the loop and increment the value of i and skip the printf statement so we get the output -- 12456789

#383 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

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

```
int a = 5;
```

```
void func(void);
```

```
void main() {
```

```
    func();
```

```
    printf("%d", a);
```

```
}
```

```
void func(void) {
```

```
    int a = 2;
```

```
printf("%d\t", a);  
}
```

- **A**
2 5
- **B**
5 2
- **C**
2 2
- **D**
5 5

Correct Answer :A

Explanation

Always program execution starts with main , In this program after main we get a function call which will redirect to the function definition . When compiler come at function definition the local variable will print output as “2” then we get \t

\t which is use for tab space (7 or 8 space depend on compiler), and when function ends cursor return where function called and execute the next line and print the output as “5”

And we get final output -- 2 5

#384 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
#include<stdio.h>

int main()

{

    int x=10; int y;

    {

        y=x++;

    }

    printf("%d",x);

}
```

- A 10
- B 11
- C 21
- D 20

Correct Answer :B

Explanation

x++ increments the value to 11.

So printf statement uses x=11.

#385 Explained Report Bookmark

Which of the following statements are correct regarding arrays?

- i) Array elements are stored in contiguous memory locations
- ii) Size of array can be mentioned anywhere in the program
- iii) The Expression arr[i] refers to i+1 element in array arr
 - A All are true
 - B i only
 - C i and iii
 - D Nothing is true

Correct Answer :B

Explanation

Here only 1 statement is right , because we cannot mention the size of the array anywhere . and the 3rd statement is also false because we only have i number of element in that array and it start from 0

int a[5]

0

1

2

3

4

It stores 5 elements only .

#386 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
{  
    int u1; float u2;  
  
    printf("%d", sizeof(u2+sizeof(u1)));  
  
}
```

- **A**
8
- **B**
4
- **C**
6
- **D**
Compilation Error

Correct Answer :B

Explanation

: According to C99 standard, the sizeof operator yields the size (in bytes) of its operand, which may be an expression or a parenthesized name of a type. The size is determined from the type of the operand. If the type of the operand is a variable length array type, then the operand is evaluated; otherwise, the operand is not evaluated and the result is an integer constant.

#387 Explained Report Bookmark

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

```
void main( )  
{  
    int a=2, b, c,d;  
  
    b=c=5;  
  
    d=++a-      --b*c%b;  
  
    printf("%d", d);  
}
```

- A
3
- B
2
- C
-1
- D
Error

Correct Answer :A

Explanation

Given expression can be written as $(++a) - (--b) * c \% b$. a will become 3 from 2 and b will become 4 from 5 due to preincrement and predecrement operators respectively. Now the expression will become $3 - 4 * 5 \% 4$ and then finally output will be 3

#388 [Explained](#) [Report](#) [Bookmark](#)

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

```
void main( )  
  
{  
  
    char b='C', c='B';  
  
    printf("%c", c+b-36);  
  
}
```

- **A**
133
- **B**
A
- **C**
a
- **D**
Garbage ASCII character

Correct Answer :C

Explanation

Given expression $c+b-36$ can be evaluated as $67+66-36$ which gives 97 and 97 is the ASCII value of a in C programming language.

#389 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
  
{  
  
    char c1='I', c2='2';  
  
    int sum=c1+c2;  
  
    printf("%c %d", sum, sum);  
  
}
```

- **A**
{ 123
- **B**
C 99
- **C**
3 51
- **D**
3 garbage

Correct Answer :A

Explanation

Given expression `c1+c2` can be evaluated as `73+50` which gives `123`. Due to `%c`, `{` will be printed in the output because `123` is the ASCII value of `{` and due to `%d`, `123` will be printed in the output.

#390 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
  
{  
  
    int product;  
  
    char ch='CH';  
  
    product("%d", product);  
  
}
```

- **A**
99
- **B**
67
- **C**
Garbage
- **D**
Error

Correct Answer :D

Explanation

Compilation errors and warnings will be generated due to multi-character constant is assigned to ch which is of char data type during initialization and due to the term product. Compilation error will be like this

[Warning] multi-character character constant [-Wmultichar]

[Warning] overflow in implicit constant conversion [-Woverflow]

[Error] called object 'product' is not a function or function pointer

[Note] declared here

#391 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )
{
    int p=1, q=2, r=3;

    float p=1.0, q=2.0, r=3.0;

    printf("%d %d %f", p, q, r);
}
```

- **A**
1 2 3
- **B**
1.0 2.0 3.0
- **C**
1 2 3.0
- **D**
Error

Correct Answer :D

Explanation

Compilation error will occur due to the redefinition of variables p, q and r. Definition means allocating memory to a variable and two memory locations cannot have same variable name.

#392 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

```
void main( )  
{  
    char ch=20;  
  
    printf("%d %d", ch>>4, ch<<2 );  
  
}
```

- **A**
180 10

- **B**
5 40
- **C**
1 80
- **D**
80 1

Correct Answer :C

Explanation

The `>>` is a right shift operator. Right shift operator is equivalent to division by 2 to the power right operand (i.e. $2^{\text{right operand}}$). Therefore `ch>>4` is evaluated as $20/16$ which gives 1 as the output. The `<<` is a left shift operator. Left shift operator is equivalent to multiplication by 2 to the power right operand (i.e. $2^{\text{right operand}}$). Therefore `ch<<2` is evaluated as $20*4$ which gives 80 as the output.

#393 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

```
void main( )  
{  
    int a=5, b=2;  
  
    printf("%d %f %f", a/b, (float)a/b,a/(float)b );  
}
```

- A
2 2.000000 2.500000
- B
2 2.5 2.5
- C
2 2.500000 2.500000
- D
none

Correct Answer :C

Explanation

The expression a/b will give 2. (float) provides an explicit type conversion of int data type to float data type. Therefore (float)a/b is evaluated as 2.500000 and a/(float)b is also evaluated as 2.500000.

#394 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

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

- A
0 1 1

- **B**
1 1 1
- **C**
0 1 0
- **D**
none

Correct Answer :A

Explanation

The given expression can be written as $k = (i-- \&& ++j) || --i$. As $i--$ will decrement value of i after evaluation and assignment and $++j$ will increment value of j before evaluation of the expression and assignment then $1 \&& 1$ gives 1 or true only. The given expression's second condition need not be evaluated due to short circuit evaluation. Therefore the second condition won't be evaluated and i will be decremented only once after evaluation and assignment. i will contain 0 and j will contain 1 and k will contain 1 because the given expression turns out to be true.

#395 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

```
void main( )  
  
{  
  
    int i=20, j=30;  
  
    i^=j;  
  
    j^=i;
```

```
i^=j;  
  
printf("%d, %d", i, j);  
  
}
```

- **A**
20, 20
- **B**
20, 30
- **C**
0,0
- **D**
30, 20

Correct Answer :D

Explanation

The `^` is a Bitwise XOR operator in C programming language. This operator is a binary operator. First of all, 20 and 30 will be converted into binary number system. `^=` means first perform Bitwise XOR between left operand and right operand and then output will be assigned to the left operand. Table of Bitwise XOR is given below

A	B	$A \oplus B$
---	---	--------------

0 0 0

0 1 1

1 0 1

1 1 0

#396 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )
{
    unsigned long val=0x12345678;
    printf("0x%x", (val>>16) | ((val& 0xffff)<<16));
}
```

- A
0x12345678
- B
0x1234
- C
0x87654321
- D
0x56781234

Correct Answer :D

Explanation

Brackets will be solved first. Left shift operator (`<<`) has higher precedence over Bitwise OR (`|`) therefore `<<` will be evaluated before `|` operator

#397 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
{  
    char ch=-3;  
  
    printf("%d %d %d", -ch, -ch+1, !ch );  
  
}
```

- A
3 -3 1
- B
3 4 0
- C
3 132 0
- D
-3 132 1

Correct Answer :B

Explanation

ch will give 3 as the output. -ch+1 will give 4 as the output and !ch means if ch is true or non-zero then make it as false or zero or if ch is false or zero then make it as true

or non-zero. ch in the given problem is -3 which is considered as true in C programming language because it is non-zero and !ch makes it false i.e. 0.

#398 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
  
{  
  
    printf("%d", 32>>2 );  
  
}
```

- A 8
- B 30
- C 128
- D 64

Correct Answer :A

Explanation

The `>>` is a right shift operator. Right shift operator is equivalent to division by 2 to the power right operand (i.e. $2^{\text{right operand}}$). Therefore $32 >> 4$ is evaluated as $32/4$ which gives 8 as the output.

#399 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

```
main()
{
    unsigned int i;
    for( i=1; i>-2;i--)
        printf(" C Aptitude");
}
```

- **A**
C Aptitude
- **B**
No output
- **C**
Compiler error
- **D**
Linker error

Correct Answer :B

Explanation

In comparison, if one operand is unsigned, then the other operand is implicitly converted into unsigned if its type is signed. Here -2 will be converted into 2 only and therefore $1 > 2$ condition will become false and no output will be produced

#400 [Explained](#) [Report](#) [Bookmark](#)

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

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

```
int main()
```

```
{
```

```
    int x;
```

```
    float y=7.0;
```

```

switch(x=y+1)

{

    case 8: printf("Right choice   "); break;

    default: printf("Oops No choice here!!!");

}

return 0;

}

```

- **A**
Error
- **B**
Oops No choice here!!!
- **C**
Right choice
- **D**
Right choice Oops No choice here!!!

Correct Answer :C

Explanation

In this program we get “Right choice” as output because x is the integer variable and in switch condition if we evaluate the expression it will give integer value which is 8 and then it match with case 8

Execution of switch condition :-

switch(x= y+1) = switch(x=7.0 +1) = switch (8.0) but because of integer variables it is considered as 8 and print the statement , after printing we get a break which is used to come out of the switch case.

#401 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

```
#include <stdio.h>

int fun(int arr[], int n)

{
    int i;

    int x = arr[0];

    for (i = 1; i < n; i++)
        x = x ^ arr[i];

    return x;
}

int main()

{
```

```
int arr[] = {12, 2, 11, 2, 2, 10, 9, 12, 10, 9, 11, 2};  
  
int val = fun(arr, 3);  
  
printf("val = %d\n", val);  
  
return 0;  
  
}
```

- A
0
- B
1
- C
12
- D
5

Correct Answer :D

Explanation

12 2 11 2 2 10 9 12 10 9 11 2

In main we call a fun in which we passed two parameters. 1st is the address of the 1st element of the array (which represents the whole array) and other is the constant value (3).

In function , x initialize with arr[0] and its value is 12

For loop start x=12 , n=3

1st iteration i = 1 , now check condition 1<3 (true), execute the depression

$x=x \wedge arr[1] \Rightarrow 12 \wedge 2$ and the 12 xor 2 give the result 14 , then it increment the value of i

x=14

2nd iteration i = 2 , 2<3 (true), execute $x= x \wedge arr[2] \Rightarrow 14 \wedge 11$ give result 5

3rd iteration i = 3 , 3<3 (false) and it return the value of x which is 5

#402 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program

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

```
char fun(void)

{
    puts(" Hello ");
    return 'A';
}

int main(void)

{
    printf("%d", sizeof(fun()));
    return 0;
}
```

- **A**
Hello A
- **B**
Hello 4
- **C**
1
- **D**
Error

Correct Answer :C

Explanation

In this program sizeof operator doesn't evaluate any expression or any printf within the function. It only consider what the fun() is returning

And the return type of fun() is char which have 1 byte size

So it will print 1 on console

#403 [Explained](#) [Report](#) [Bookmark](#)

What can not be used for initializing arrays?

- A
''
- B
“ ”
- C
{ }
- D
[]

Correct Answer :D

Explanation

:when we initialize the array we do not use these brackets[]

Eg int a[] = { 'a', 'b' };

#404 [Explained](#) [Report](#) [Bookmark](#)

A[j] is not the same as _____.

null

- A
*(A+j)
- B
*(j+A)
- C
j[A]
- D
A+j

Correct Answer :D

Explanation

In c we can access the value of array in three ways which are following below :-

1) $*(A+j)$

2) $*(j+A)$

3) $j[A]$

This all will give the value

#405 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )
{
    int a[ ]={20, 30, 40, 50, 60}, *j=a;
    j+=3;
    printf("%d", *j);
}
```

- A
50
- B
20

- **C**
40
- **D**
Compilation error

Correct Answer :A

Explanation

20

30

40

50

60

In this program j gets the whole array , which means j is pointing the starting index of an array . now in next line $j+=3$ means $j= j+ 3$ which point the $a[3]$ index and because of * we can get the value of index 3

And get 50 as output

#406 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )
```

```
{
```

```
char p[5]={0,0,65,66};  
  
    printf("%d %d", p[1], p[4]);  
  
}
```

- **A**
0 66
- **B**
0 0
- **C**
0 garbage
- **D**
Error

Correct Answer :B

Explanation

In this program , it print the value of p[1] and p[4] on console

0	1	2	3	4
0	0	65	66	0

But here p[4] is not given in the program sop it initialize with 0 and p[1] is also 0

So we get 0 0 as output

#407 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
{  
    char a[12]={ 'S', 'u', 'n',  
'C', 'o', 'm', 'p', 'u', 't', 'e', 'r' };  
  
    printf("%s", a);  
}
```

- **A**
Sun
- **B**
SunComputer
- **C**
Sun Computer garbage
- **D**
Error

Correct Answer :B

Explanation

S u n C o m p u t e r \0

It will print SunComputer on console because here %s used which is use to string

#408 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
void main( )  
  
{  
  
    int a[ ]={10,20,30,40,50}, j;  
  
    for(j=0; j<5; j++)    printf("%d", *a);  
  
}
```

- **A**
10 10 10 10 10
- **B**
10 20 30 40 50
- **C**
Garbage value printed 5 times
- **D**
Error : L value required

Correct Answer :A

Explanation

In this program the loop starts from $j= 0$, then it checks the condition $1 < 5$ (true), when condition true it comes inside the loop and executes the print statement . here we get $*a$ which point the first element of the array it means it prints 10 on the console . and increment the value of j by 1

2nd iteration , now $j= 1$, $1 < 5$ (true) and again it prints the same value in the print statement which is 10 (because here we didn't increment the position). And then increments the value of j by 1.

3rd iteration , now $j= 2$, $2 < 5$ (true) and again it prints the same value in the print statement which is 10 And then increments the value of j by 1.

4th iteration , now $j= 3$, $3 < 5$ (true) and again it prints the same value in the print statement which is 10 And then increments the value of j by 1.

5th iteration , now $j= 4$, $4 < 5$ (true) and again it prints the same value in the print statement which is 10 And then increments the value of j by 1.

5th iteration , now $j= 5$, $5 < 5$ condition false and it comes out of the loop.

And we get output 10 10 10 10 10 on console

#409 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
. void main( )  
{  
    char a[6*3/2]={ 'a' , 'b' , 'r' , 'd' , 'c' };  
    printf("%c", a[3] );
```

}

- A
r
- B
d
- C
c
- D
Error

Correct Answer :B

Explanation

In this program, `char [6 *3/2]` will execute, here * and / have same precedence so we go with associativity and it means it execute left to right
 $6*3 = 18$ and $18 / 2 = 9$

It means it have 9 elements in an array like this

0 1 2 3 4

a b r d c

And in this program we have to print the character at 3rd position which is d

#410 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
. void main( )  
{  
    int    n[3][3]={2,4,6,8,5,3,5,1}, *ptr ,i;  
  
    ptr=&n[1][0];  
  
    for(i=0;i<=3;i++) printf("%d", *(ptr+i));  
}
```

- **A**
8 5 3 5
- **B**
5 3 5 1
- **C**
3 6 8 5
- **D**
6 3 garbage garbage

Correct Answer :A

Explanation

Two-dimensional array store like this

0 1 2

2 4 6

8 5 3

5 1 0

Here ptr is pointer type variable ans in program we assign the address to ptr

ptr = &n[1][0] it means address of 1st row and 0 column (and value at this place is 8).

Now for loop execution start

1st iteration i = 0 , now it check condition $0 \leq 3$ (true), it come inside the loop and print the $*(\text{ptr}+i) = *(\text{ptr}+0)$, here ptr has an address of 1st row and 0th col, it will print the value of this address which is 8 . and then it execute the increment part of the loop by 1

2nd iteration, now i = 1, condition $\rightarrow 1 \leq 3$ (true) , print $*(\text{ptr}+1)$ here it show the next address and print the value 5 and increment the value of i by 1

3rd iteration, now i = 2 , condition -> $2 \leq 3$ (true) , print *(ptr+2) here it show the next address and print the value 3 and increment the value of i by 1

4th iteration, now i = 3, condition -> $3 \leq 3$ (true) , print *(ptr+3) here it show the next address and print the value 5 and increment the value of i by 1

5th iteration , now i =4 , condition -> $4 \leq 3$ (false) not it come out of the loop and we get our output as 8 5 3 5

#411 Explained Report Bookmark

What is the output of the following program:

```
void main( )  
  
{  
  
    int s[2][ ]={3,4,5,6,7,8,9,10};  
  
    printf("%d", s[1][2] );  
  
}
```

- A
8
- B
9
- C
6
- D
Error

Correct Answer :D

Explanation

Here we get error at line number 2 because here it does not mention the column number and get the error “array type incomplete”.

Correct way to declare and initialize the two -dimensional array is -- int s[2][4]={3,4,5,6,7,8,9,10};

Nopw it will give the output 9

#412[Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program:

```
. void main( )  
  
{  
  
    int M[12];    M[0]= -12;    M[11]= -12;  
  
    printf("%d %d %d", * (M+0), * (M+11), *M );  
  
}
```

- **A**
-12 12 -12
- **B**
-12 -12 -12
- **C**
-12 -12 12

- D
0 11 garbage

Correct Answer :B

Explanation

In this program only M[0] and M[11] is initialize with some value which is -12

Other places will initialize with 0 if it is not initialized by the programmer.

	0	1	2	3	4	5	6	7	8
9	10	11							

-12	0	0	0	0	0	0	0	0	-12
-----	---	---	---	---	---	---	---	---	-----

In printf statement we have to print 3 value which are

$*(M+0)$ means value of index 0 which is -12

$*(M+11)$ means value of index 11 which is also -12

$*M$ it indicate the value of base address which is -12

So we get -12 -12 -12 as output of the program

What will be the output of the following C code?

```
#include <stdio.h>

char * f();

char a = 'a';

int main(int argc, char *argv[])
{
    char *temp = f();
    printf("%%, temp);

    return 0;
}

char *f()
{
    return &a;
}
```

- A
%
- B
%a

- **C**
%%
- **D**
a

Correct Answer :A

Explanation

To print % on the screen in C programming language, two times %% within double quotes should be written in a program. Here is the same scenario.

#414Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int 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);  
  
return 0;  
  
}
```

- **A**
41,44,300
- **B**
44,41,300
- **C**
44,44,300
- **D**
44,43,300

Correct Answer :B

Explanation

Because `char` is by default `signed` declared that means the range of the variable is

-127 to +127>

your value is overflowed. To get the desired value you have to declared the `unsigned` modifier. the modifier's (`unsigned`) range is:

0 to 255

to get the the range of any data type follow the process `2bit` example `char`is 8 bit length to get its range just $2^{(power)} - 1$.

#415 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    char result,str[]="\0Ccatpreparation";
    result=printf("%s  ",str);
    if(result)
        printf("Test 1 ");
}
```

```
else

    printf("Test 2 ");

return 0;

}
```

- **A**
Ccatpreparation Test 1
- **B**
Ccatpreparation Test 2
- **C**
Test 1
- **D**
Test 2

Correct Answer :C

Explanation

A blank space is printed on the screen due to a blank space within quotes and printf() function will return 1 and it is assigned to result. result contains 1 and if condition turns out to be true and due to this first printf() will be executed.

#416Explained Report Bookmark

What will be the output of the following C code?

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

```
#define sq(x) x + 1 * x - 1

int main(void)

{
    int a = 5, b = 0;

    b = sq(a);

    printf("\n a=%d b=%d\n ", a, b);

    return 0;
}
```

- **A**
a = 5 b = 9
- **B**
. a = 5 b = 25
- **C**
a = 5 b = 24
- **D**
. a = 5 b = 29

Correct Answer :A

Explanation

x will be replaced by a and before compilation macro expansion is done and due to this in place of sq(a), a+1*a-1 will be written.

#417 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int a=16, n=0;

    while (n<=~(~a) )
    {
        n++;
    }

    a=n;

    printf("%d", ~a);

    return 0;
}
```

- A
18
- B
-17
- C
-18
- D
17

Correct Answer :C

Explanation

The \sim is a Bitwise NOT operator. Bitwise NOT is a unary operator. Its job is to complement each bit one by one. Result of NOT is 0 when bit is 1 and 1 when bit is 0. Table of Bitwise NOT is shown below:

A

$\sim A$

0

1

1

0

#418 Explained Report Bookmark

What will be the output of the following C code?

```
#include <stdio.h>

int main()
{
    int a=2;

    int b=a;

    switch(b)

    {
        case a:
            printf("Pune\n");

        case 3:
            printf("Karad\n");
            break;

        default:
            printf("Mumbai\n");
    }
}
```

```
        break;  
  
    }  
  
    printf("Exit from switch");  
  
    return 0;  
  
}
```

- **A**
Karad
- **B**
Pune Karad
- **C**
Mumbai
- **D**
Compolation Error

Correct Answer :D

Explanation

Compiler error will be like this

[Error] case label does not reduce to an integer constant.

#419 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
#include <stdio.h>

int main()

{

    int a=10,b=2;

    int *pa=&a, *pb=&b;

    printf("value = %d", *pa/*pb);

    return 0;

}
```

- **A**
5
- **B**
5.0
- **C**
Error
- **D**
None of the above

Correct Answer :C

Explanation

Compiler error will be like this

[Error] unterminated comment.

[Error] expected ')' at end of input.

[Error] expected declaration or statement at end of input.

#420 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( ) {  
    int ary[2];  
  
    ary[-2]=2;  
  
    printf("%d", ary[2-4] );  
  
}
```

- A
1
- B
2
- C
3

- D
4

Correct Answer :B

Explanation

: In this program printf will evaluate the expression and print the value of arr[-2] which is 2

#421 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
void main( )  
  
{  
  
    char dac[ ];  
  
    dac[0]=65;  
  
    printf("%c", dac[0] );  
  
}
```

- A
65
- B
a

- **C**
A
- **D**
Error

Correct Answer :D

Explanation

In this program we get error, because the length of array is not specified at line number 2 and we get error “array size missing” on console.

Correct syntax of array declaration is :-

Char[6]; or char a;

#422 [Explained](#) [Report](#) [Bookmark](#)

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

```
. int size=5;

void main( )

{
    int Array[size+1];

    printf("%d", sizeof(Array) );
}

(assume size of int = 2byte)
```

- A
2
- B
6
- C
12
- D
Error

Correct Answer :C

Explanation

: int get 2 byte and the length of an array is 6 so the size is 12

Int array[size + 1] → int array[5+1] → int array[6]

#423 [Explained](#) [Report](#) [Bookmark](#)

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

```
void main( )  
  
{  
  
    int A[2]= { '1', '2' } ;  
  
    A[ '1' - '2' ]=50 ;  
  
    printf("%d", 1[A] ) ;
```

```
}
```

- A
1
- B
2
- C
50
- D
Error

Correct Answer :C

Explanation

In this program , we get 50 as output because $A['1'-'2']=50$; at this line it will get ASCII value of 1 and 2 and we get $A[1] = 50$

So it will print 50 on console

#424 [Explained](#) [Report](#) [Bookmark](#)

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

```
void junk(int i, int *j)
```

```
{
```

```
i=i*i;
```

```
*j= *j * *j;
```

```

}

void main( )

{

    int i=-5, j=-2;

    junk(i, &j);

    printf("i=%d j=%d", i, j);

}

```

- **A**
i=25 j=4
- **B**
i=-5 j=-2
- **C**
i=-5 j=4
- **D**
i=5 j=4

Correct Answer :C

Explanation

In this program $i = -5$ and $j = -2$, here we get a function call. In this we pass two parameter and both parameters are different, first we use “call by value” which means the value of actual parameter(i) will be copied to formal parameter ($int i$) and these 2 parameter store value in different place

The 2nd is call by reference , In this actual (`&j`) and formal (`*j`) parameter refer to the same memory location,

In function definition `i = i * i` // it will evaluate but changes will remain in function definition

`j= *j * *j -> j = -2 * -2 = -4` it will change on original address

#425 [Explained](#) [Report](#) [Bookmark](#)

`typedef int (*PFI)(char *, char *)` creates:

null

- A type PFI, for pointer to function (of two `char *` arguments) returning `int`
- B type PFI, function (of two `char *` arguments) returning `int`
- C type PFI, for pointer
- D Error

Correct Answer :A

Explanation

`typedef` give freedom to user by allowing to adds a new name for some existing type which and here PFI is a pointer to function which has two arguments and the return type of this function is `int`

#426 [Explained](#) [Report](#) [Bookmark](#)

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

```
#include <stdio.h>

int
main ()
{
    typedef float f;

    static f *fptr;

    float fval = 90;

    fptr = &fval;

    printf ("%f\n", *fptr);

    return 0;
}
```

- **A**
0
- **B**
90.000000
- **C**
9
- **D**
90

Correct Answer :B

Explanation

here we create our own data type f which is of float type and fptr is a pointer-type variable which store the value of fval which is float type so we get 90.000000

#427 [Explained](#) [Report](#) [Bookmark](#)

What are the Benefits of Function Pointers?

- **A**
Function pointers provide a way of passing around instructions for how to do something
- **B**
You can write flexible functions and libraries that allow the programmer to choose behavior by passing function pointers as arguments
- **C**
All of the Above.
- **D**
None of the above

Correct Answer :C

Explanation

In c function pointer is used as a variable which store the address of a function and it return the pointer

#428 [Explained](#) [Report](#) [Bookmark](#)

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

```
#include<stdio.h>

int main()

{

int mat3[][] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

int r, c;

printf("\n Enter Elements of Matrix :: \n");

for (r = 0; r < 3; r++)

{

for (c = 0; c < 3; c++)

{

scanf("%d", &mat3[r][c]);

}

}

for (r = 0; r < 3; r++)

{

for (c = 0; c < 3; c++)

{
```

```
    printf("%d", mat3[r][c]);\n\n}\n\nreturn 0;\n}\n\n}
```

- **A**
Compile time error
- **B**
1 2 3 4 5 6 7 8 9
- **C**
Garbage Values
- **D**
9 8 7 6 5 4 3 2 1

Correct Answer :A

Explanation

because on second line we did not mention the length

The correct way is int mat3[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

#429 [Explained](#) [Report](#) [Bookmark](#)

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

```
#include<stdio.h>

void sample(int**);

int main()

{

int arr[3][4]={{1,2,3,4},{4,3,2,8},{7,8,9,0}};

int *ptr;

ptr=&arr[0][0];

sample(&ptr);

return 0;

}

void sample(int **p)

{

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

}
```

- **A**
1
- **B**
compile time error

- **C**
Run time error terminated with -1
- **D**
2

Correct Answer :A

Explanation

This two dimensional array stores like this

1 2 3 4

4 3 2 8

7 8 9 0

And the address of row 0 col 0 store in the ptr(pointer variable) and here we pass the address of ptr to the function which prints the value 1 on console

#430 Explained Report Bookmark

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

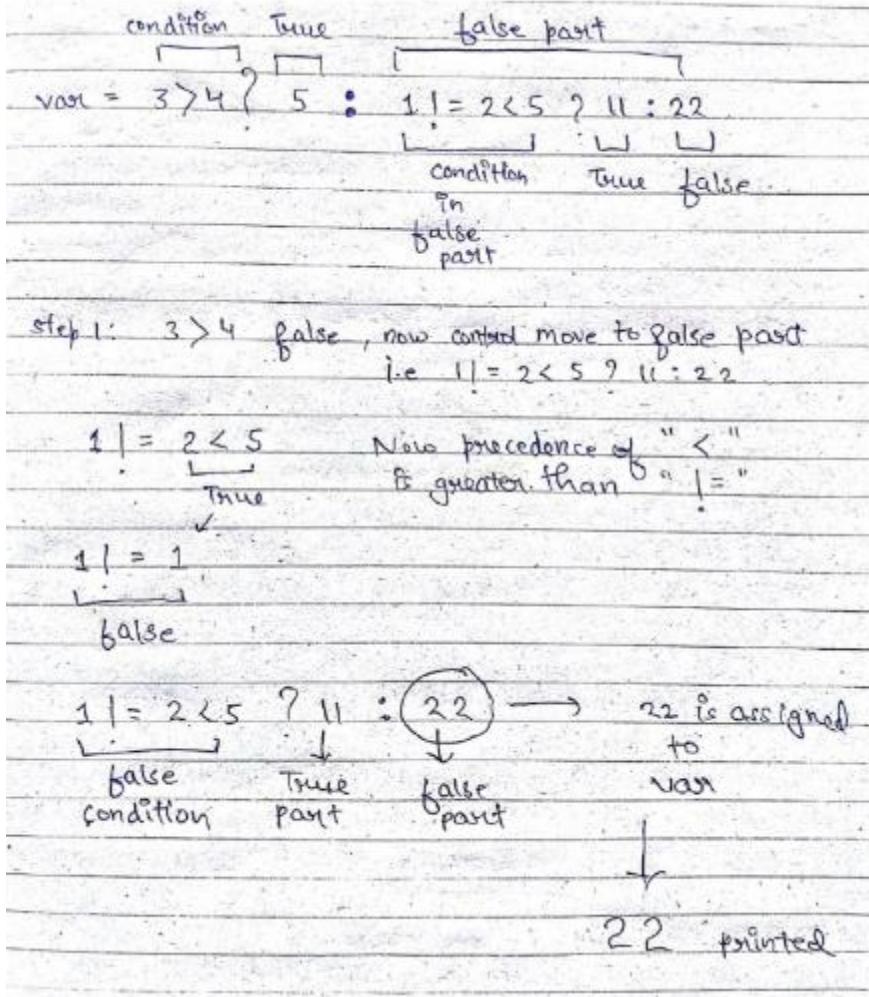
```
#include <stdio.h>
```

```
int main()  
{  
    int var;  
  
    var=3>4?5:1!=2<5?11:22;  
  
    printf("%d",var);  
  
    return 0;  
}
```

- **A**
22
- **B**
11
- **C**
5
- **D**
1

Correct Answer :A

Explanation



#431 Explained Report Bookmark

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

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

```
int main()
```

```
{
```

```
    int one , two =1 , three;
```

```
if(three ==(two ==0)); one = 5; two =3;  
  
printf("%d%d%d",three,two,one);  
  
return 0;  
}
```

- **A**
1 1 5
- **B**
0 3 garbage value
- **C**
0 1 5
- **D**
0 3 5

Correct Answer :D

Explanation

```

int one, two = 1, three;           multiple line
if (three = (two == 0)); one = 5; two = 3; ✓
                                ↓
                                if is terminated with empty body ✓
if (three = (two == 0))
{
    3   ↓   1 == 0
one = 5;   false
two = 3;   ↓
            0
if (three = 0)
↓
assigning 0 to variable three ✓
if (0) → false
}
Now three = 0 [ assigning ]
one = 5;
two = 3;
printf(" three, two, one )
      ↓   ↓   ↓
      0   3   5

```

#432 Explained Report Bookmark

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

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

```
int main()
```

```
{
```

```

int y = 1;

if(y<0)

    if(y>0);

        printf("welcome");

else

    printf("ccatpreparation");

return 0;

}

```

- A welcome
- B ccatpreparation
- C no output
- D compile time error

Correct Answer :D

Explanation

Actually when IF statement is written without parenthesis then the very first statement is considered as a part of the body of IF statementTherefore 2nd if is considered as a nested if for 1st IF

statement.....Now because of ; the second if is terminated.....Now the whole IF block is completed.....Now there is a print statement which is not related to IF block.....Then else block is writtenCompiler error will be mismatch else statement because else should always be written after IF block but printf statement is written between IF and ELSE blockHence compile time error

#433 [Explained](#) [Report](#) [Bookmark](#)

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

```
#include <stdio.h>

int main()

{
    int one ,two =1 , three;

    if(three ==(two ==0)) one = 5; two =3;

    printf("%d%d%d",three,two,one);

    return 0;
}

• A
  0 3 5
• B
  1 1 5
```

- C
0 3 -32767
- D
0 3 32767

Correct Answer :D

Explanation

if condition is evaluated as false

two == 0 --> return false (0)

threee = 0

so if condition block will not executed thats why one is initialize with max range

so output 0 3 32767

#434 [Explained](#) [Report](#) [Bookmark](#)

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

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

```
int main()
```

```
{
```

```
    int itr =0;
```

```
for(;itr++<0;printf("%d",itr));  
  
    printf("%d",itr);  
  
  
  
  
return 0;  
  
}
```

- **A**
0
- **B**
1
- **C**
Nothing
- **D**
None of the above

Correct Answer :B

Explanation

ccatpreparation.com

```
for( ; i++ < 0 ; printf("%d", i)) ;
```

empty initialization condition post increment that means value of i is used first then it is incremented ; means empty for loop body

$i < 0$ false

control moves out of for loop body
Now since $i++$ is a post increment
its value is now incremented by 1
 $i = 1$ ✓

Now the printf statement after for block is executed and it prints value of i that is 1

#435 Explained Report Bookmark

What is the output of the following code snippet?

```
#include<stdio.h>

main()
{
    int const a = 5;
    a++;    printf("%d", a);
```

}

- A
5
- B
5
- C
Runtime error
- D
Compile error

Correct Answer :D

Explanation

In this program we declare a as a constant variable and in the next line we are trying to post increment the variable which is not possible. in this program it gives a compile time error because a is a read-only variable .

#436 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main()

{
    int x = 65, *p = &x;
```

```
void *q=p;  
  
char *r=q;  
  
printf("%c",*r);  
  
}
```

- A Garbage character.
- B A
- C 65
- D Compile error

Correct Answer :B

Explanation

In this program we declare a variable x and initialize with 65 and the address of x is store in pointer variable p ,

in next line we just assign the value of p into pointer variable q (that means q also hold the address of x) and

in next line we again assign the value of q into pointer variable r which is of character type and also use %c (which is used for char) in print statement

so , the ans is A because the ascii value of A is 65

#437 Explained Report Bookmark

What is the output of the below code snippet?

```
#include<stdio.h>

main() {
    char s[]="hello",
        t[]="hello";
    if(s==t) {
        printf("equal strings");
    }
}
```

- A Equal strings
- B Unequal strings
- C No output
- D Compilation error

Correct Answer :C

Explanation

In this program if we can not compare both strings , and if we do so then the base address of both strings will compare which is different and that's why if condition becomes false and it will not print anything on console.

#438 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
#include<stdio.h>

main()  {

int a = 5, b = 3, c = 4;

printf("a = %d, b = %d\n", a, b, c);

}
```

- A
a=5, b=3
- B
a=5, b=3, c=0
- C
a=5, b=3, 0
- D
compile error

Correct Answer :A

Explanation

In this program we use 2 %d then the first two variable values will print on screen , because here there is no conversion character like %d is written for variable c .

So only a and b will print on console.

#439 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
#include<stdio.h>

main()
{
    int a = 1;

    float b = 1.3;

    double c;

    c = a + b;

    printf("%.2lf", c);

}
```

- A
2.30
- B
2.3

- **C**
Compile error
- **D**
2.0

Correct Answer :A

Explanation

In this program value of c print on console which is 2.30, because here we use %.2lf which show that we have to take 2 values after decimal

So, $c = 1 + 1.3 = 2.3$

But we take 2 values after decimal so the final output is 2.30

#440 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    enum { india, is=7, GREAT } ;
    printf("%d %d", india, GREAT) ;
}
```

- A
0 1
- B
0 2
- C
0 8
- D
Compile error

Correct Answer :C

Explanation

In enum when we start from 0 so india will get 0 and after this we initialize is with 7 and then it next will continue this initialization so Great will get 8 and our final output is 0 8

#441 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following code snippet?

```
#include<stdio.h>

main() {     char c = 'A'+255;         printf("%c", c); }
```

- A
A
- B
B
- C
@
- D
Compile error

Correct Answer :C

Explanation

In c range of character is 0-255 and suppose if we add 1 the it become 256 but here the variable is character type so it become 0

In this program `c= 'A'+255 = 65 +255 = 320` and it repeat the cycle and we get `c=64` which is @ in ASCII table

Output is @

#442 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following code snippet?

```
#include<stdio.h>

main() {    short unsigned int i = 0;           printf("%u\n", i--
) ; }
```

- A
0
- B
Compile error
- C
65535
- D
32767

Correct Answer :A

Explanation

In this program we initialize i with 0 and in print statement it is post decrement that means first use the value then decrement by 1 so our output will be 0

#443 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
#include<stdio.h>

main() {    unsigned x = 5, y=&x, *p = y+0;

printf("%u", *p); }
```

- A Address of x
- B Address of y
- C Address of p
- D 5

Correct Answer :D

Explanation

Turbo C Compilation

Here,

we assign 5 to variable x and store the address in y variable and in the next line we add with 0

which will remain the same and print the value of x on console .

#444 [Explained](#) [Report](#) [Bookmark](#)

What is your comment on the below C statement?

```
signed int *p=(int*)malloc(sizeof(unsigned int));
```

- A Improper type casting
- B Would throw Runtime error
- C Memory will be allocated but cannot hold an int value in the memory
- D No issue with statement

Correct Answer :D

Explanation

The size of int and unsigned data type is same therefore there is no problem with this statement

```
signed int *p=(int*)malloc(sizeof(unsigned int));
```

#445 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following code snippet?

```
#include<stdio.h>

main()

{
    int x = 5;

    if(x==5)
    {
        if(x==5) break;

        printf("Hello");
    }

    printf("Hi");
}
```

- A
Compile error
- B
Hi
- C
HelloHi

- D
Hello

Correct Answer :A

Explanation

In this use break statement in if condition. Break statements should be used only in loops.

#446 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following code snippet?

```
#include<stdio.h>

main() {    int x = 5;        if(x==5)      {            if(x==5)
printf("Hello");    }    printf("Hi"); }
```

- A
HelloHi
- B
Hi
- C
Hello
- D
Compiler error

Correct Answer :A

Explanation

In this program we assign 5 to x variable and in if statement we are not comparing , here we assign the value which means in if we get non-zero value which is true and execute the if statement and we get final output - HelloHi

#447 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
#include<stdio.h>

main() { for();printf("Hello"); }
```

- A Infinite loop
- B Prints “Hello” once.
- C No output
- D Compile error

Correct Answer :D

Explanation

The syntax of for loop is :-

```
for(initialization ; condition; increment/decrement )
```

```
{
```

```
// statements
```

}

Empty loop is not allowed

#448[Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet?

```
#include<stdio.h>

main() {    for(1;2;3)        printf("Hello"); }
```

- A
Infinite loop
- B
Prints “Hello” once.
- C
No output
- D
Compile error

Correct Answer :A

Explanation

In this program we use non-zero value at the place of initialization , condition and increment so it will run infinite time . because we always get true condition.

#449[Explained](#) [Report](#) [Bookmark](#)

int x=~1; What is the value of 'x'?

- A
1
- B
-1
- C
2
- D
-2

Correct Answer :D

Explanation

Here int x= ~1 , here we get the value of x by a formula which is

$$\sim n = -n-1$$

So here, we get ~1, then x = -1-1 which is equal to -2

#450 Explained Report Bookmark

What is the output of the following program?

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

```
void f() {
```

```
    static int i;
```

```
    ++i;
```

```
    printf("%d", i);
```

```
}
```

```
main() {      f();      f();      f(); }
```

- A
1 1 1
- B
0 0 0
- C
3 2 1
- D
1 2 3

Correct Answer :D

Explanation

First program start it's execution from main and we get 3 function call , and in function f we used static variable which preserve it's value even after they are out of their scope

And when it is not initialized by default it get 0 for int

In 1st function call it initialize with 0 and in next line we increment the value of i by 1 and print on console

In the 2nd function call we get the previous value which is 1 and in the next line it increments by 1 , now i become 2 and printed on the console.

In 3rd function call we get 3 on console with same procedure

So the final output is 1 2 3

#451 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following code snippet?

```
#include<stdio.h>

main() {    int *p = 15;    printf("%d", *p); }
```

- A
15
- B
Garbage value
- C
Runtime error
- D
Compiler error

Correct Answer :C

Explanation

In this program we get runtime error - segmentation fault

A segmentation fault occurs when a program attempts to access a memory location that it is not allowed to access, or attempts to access a memory location in a way that is not allowed (for example, attempting to write to a read-only location, or to overwrite part of the operating system).

In this program we haven't give memory location to pointer p and directly assign it a value using * operator.

If we write int *p = &x and then assign any value to *p then it is correct.

#452 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {      register int x = 5;

    int *p;      p=&x;      x++;      printf("%d",*p); }
```

- A
Compile error
- B
6
- C
5
- D
Garbage value

Correct Answer :A

Explanation

It will give an error - address of register variable x

Registers are faster than memory to access, so the variables which are most frequently used in a C program can be put in registers using the register keyword. The keyword register hints to the compiler that a given variable can be put in a register.

If you use & operator with a register variable then the compiler may give an error, because when we say a variable is a register, it may be stored in a register instead of memory and accessing address of a register is invalid.

#453 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {    int i = 13, j = 60;        i ^= j;    j ^= i;    i ^=
j;    printf("%d %d", i, j); }
```

- A
73 73
- B
60 13
- C
13 60
- D
60 60

Correct Answer :B

Explanation

In this program we use Bitwise Xor which will give 1 when we get either a is 1 or b is 1 and if both are 1 then output is 0

Here i= 13 and j=60 , when we perform bitwise operation we use binary form

i = i ^ j = 13 ^ 60

13 - 0 0 1 1 0 1

60- 1 1 1 1 0 0

 ^

Ans 49 - 1 1 0 0 0 1, now i=49

j= j ^ i = 60 ^ 49

60- 1 1 1 1 0 0

49- 1 1 0 0 0 1

 ^

We get j =13 = 0 0 1 1 0 1

And in last again we do i = i ^ j ,

49- 1 1 0 0 0 1

13 - 0 0 1 1 0 1

^

We get i= 60

So our final output is 60 13.

#454 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

void f()

{

printf("Hello\n");

}

main() { ; }
```

- A
No output
- B
Error, as the function is not called
- C
Error, as the function is defined without its declaration
- D
Error, as the main() function is left empty

Correct Answer :A

Explanation

in the given Program we declared the function but in main block we are not calling any function so there is no out put

#455 [Explained](#) [Report](#) [Bookmark](#)

Which operator is used to continue the definition of macro in the next line?

- A
#
- B
##
- C
\$
- D
\

Correct Answer :D

Explanation

The backslash character ('\') is the line continuation character.

#456 [Explained](#) [Report](#) [Bookmark](#)

What is the size of 'int'?

null

- A
2
- B
4

- C
8
- D
Compiler dependent

Correct Answer :D

Explanation

The size of int is compiler dependent. It depends on both processors (more specifically, ISA, instruction set architecture, e.g., x86 and x86-64) and compilers including programming model. For example, in 16-bit machines, sizeof (int) was 2 bytes. 32-bit machines have 4 bytes for int.

#457 [Explained](#) [Report](#) [Bookmark](#)

What is the value of 'y' for the following code snippet?

```
#include<stdio.h>

main()
{
    int x = 1;

    float y = x>>2;

    printf( "%f", y );

}
```

- A
4
- B
0.5
- C
0
- D
1

Correct Answer :C

Explanation

Bitwise Right Shift Operator

It is denoted by **>>** Bit Pattern of the data can be shifted by specified number of Positions to Right. When Data is Shifted Right , leading zero's are filled with zero. Right shift Operator is Binary Operator [Bi – two]

#458 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
main()
```

```
{
```

```
float t = 2;
```

```
switch(t)
```

```
{  
  
    case 2: printf("Hi");  
  
    default: printf("Hello");  
  
}  
  
}
```

- A
Hi
- B
HiHello
- C
Hello
- D
Error

Correct Answer :D

Explanation

[Error] switch quantity not an integer.

The expression used in switch must be integral type (int, char and enum). Any other type of expression is not allowed.

#459 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
include<stdio.h>
```

```
main()

{
    int i = 1;

    while(++i <= 5)

        printf("%d ",i++);

}
```

- A
1 3 5
- B
2 4
- C
2 4 6
- D
2

Correct Answer :B

Explanation

2 4 will be the output because only two times loop's statements will be executed and at third time the condition will become false.

#460 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main()
{
    int i = 1;

    while( i++<=5 )

        printf("%d ",i++);

}
```

- A
1 3 5
- B
2 4
- C
2 4 6
- D
2

Correct Answer :C

Explanation

: 2 4 6 will be the output because only three times loop's statements will be executed and at fourth time the condition will become false.

#461 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main()
{
    int i = 1;

    while(i++<=5);

    printf("%d ",i++);
}


```

- A
4
- B
7
- C
2 6
- D
2 4

Correct Answer :B

Explanation

After while statement we have semicolon so it will increment the value of i till 7 then it print final value

#462 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main()

{
    int x = 1;

    do
        printf("%d ", x);

    while(x++<=1);

}
```

- A
1
- B
1 2
- C
No output
- D
Compile error

Correct Answer :B

Explanation

In do-while statement do block is executed first time without checking the condition in while

so it print 1 first time then inside while loop we are increment x++ so value is 1 which satisfy the condition again do block executed print 2

this time value of x is 2 inside while block condition un satisfy it will exit

#463 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main ()
{
    int a=1, b=2, *p=&a, *q=&b, *r=p;

    p = q; q = r;

    printf("%d %d %d %d\n",a,b,*p,*q);

}
```

- **A**
1 2 2 1
- **B**
2 1 2 1
- **C**
1 2 1 2

- D
Compile error

Correct Answer :A

Explanation

In this program we assign value to variable a and b and also assign the address of a and b into pointer variable p and q , after this we are trying to swap the value of p and q

a

1

1001

b

2

1003

Here we take one more pointer variable r and assign the value of p (which is 1), in next line we assign value of q into p (now p hold value 2) and in last value of r assign to q, and now q hold value 1

So the output is 1 2 2 1

#464 Explained Report Bookmark

In the given below code, if a short int value is 5 bytes long, then how many times the while loop will get executed?

```
#include<stdio.h>

int main ()
{
    int j = 1; while(j <= 300) {
        printf("%c %d\n", j, j); j++; }
    return 0;
}
```

- A Unlimited times
- B 0 times
- C 300 times
- D 5 times

Correct Answer :C

Explanation

In this program the while loop will run 300 times . In c < means less so in while loop the condition is

```
while( j <= 300)
```

So it will run 300 times

#465 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
int main()
{
    int x; float y;
    y = x = 7.5;
    printf("x=%d y=%f", x, y);

    return 0;
}
```

- A
7 7.000000
- B
7 7.500000
- C
5 7.500000
- D
5 5.000000

Correct Answer :A

Explanation

In this program we get value of x =7 and y = 7.000000 , because in this program first we assign value 7.5 to variable x and it is integer type variable so it will take only 7 now the value of x assign to y and y is float variable so it will give 7.000000 as output

Assignment operators work right to left .

#466 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

void f(int a[])
{
    int i;
    for(i=0; i<3; i++)
        a[i]++;
}

int main() {
    int i,a[] = {10, 20, 30};
```

```
f(a); for(i=0; i<3; ++i) {  
  
    printf("%d ",a[i]);  
  
}  
  
}
```

- A
10 20 30
- B
11 21 31
- C
Compile error
- D
Runtime error

Correct Answer :C

Explanation

In this program we get error at condition in for loop . here we use < instead of < so it will give an error . if we use condition like

`for(i =0; i< 3; i++)`

Then our output will be 11 21 31

#467 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

int main()

{

    int x = 3;

    x += 2; x += 2; printf("%d", x);

    return 0;

}
```

- A
2
- B
5
- C
7
- D
Compile error

Correct Answer :A

Explanation

In this program we get $x= 3$, and in next line is $x= x+2$ so now $x= 5$

But after this value +2 again assign to variable x

So it will give output 2

x+=2 and x = +2 both are different

x+=2 means x= x+2, but x =+2 means we are assigning the value

#468Explained Report Bookmark

What is the Correct Syntax of the Function Pointer?

- **A**
int (*fun[4])(int, int);
- **B**
int (*fun)(int, int);
- **C**
double (*ope)(int, int);
- **D**
Option 1 and option 2 both

Correct Answer :D

Explanation

int (*fun[4])(int, int) ==fun is an array of 4 pointer to function which takes 2 arguments of type int returning integer value

int (*fun)(int, int); == fun is a function pointer which takes two integers as input arguments and returns an integer value.

#469Explained Report Bookmark

What we will not do with function pointers?

- **A**
deallocation of memory
- **B**
allocation of memory

- **C**
 - . both A & B
- **D**
 - none of the mentioned

Correct Answer :C

Explanation

A function pointer is a variable that stores the address of a function

It executes a block of code, but it will not allocate and deallocate the memory.

#470 [Explained](#) [Report](#) [Bookmark](#)

Predict the output for the following code

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

- **A**

- **B**
42
- **C**
36
- **D**
30

Correct Answer :B

Explanation

In this program we get more than one operator so we will solve the expression according to its precedence table

Here postfix has higher precedence than prefix and then we multiply the values

$x = ++p * 6$, then $x = 7 * 6 = 42$

#471 [Explained](#) [Report](#) [Bookmark](#)

Predict the output for the following code

```
void main( )  
  
{  
  
int p=4;  
  
printf("%d", ++p*p++*++p) ;  
  
}
```

- A
175
- B
252
- C
216
- D
none

Correct Answer :D

Explanation

here we also get more than one operator and we solve according to precedence table

So in print statement first postfix value get updated then prefix

$$6 * 5 * 7 = 210$$

#472 [Explained](#) [Report](#) [Bookmark](#)

Linker generates ____ file.

null

- A
Object code
- B
Executable code
- C
Assembly code
- D
None of the above.

Correct Answer :B

Explanation

Linker is a program in a system which helps to link object modules of program into a single object file. It performs the process of linking. Linker are also called link editors. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also links a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader.

Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader. Linking is performed at the last step in compiling a program.

Source code -> compiler -> Assembler -> Object code -> Linker -> Executable file -> Loader

#473 [Explained](#) [Report](#) [Bookmark](#)

Compiler generates ____ file.

null

- A
Executable code
- B
Executable code

- **C**
Assembly code
- **D**
None of the above.

Correct Answer :B

Explanation

Source Code -- > Compiler --> Object Code

Source code is supposed to contain meaningful variable names and helpful comments that are intended only to be read by humans. A piece of software called a "compiler" must convert source code to object code before the program described in the source code can be executed.

#474 [Explained](#) [Report](#) [Bookmark](#)

Special symbol permitted with in the identifier name.

- **A**
\$
- **B**
@
- **C**
- **D**
. (dot)

Correct Answer :C

Explanation

Rules for an Identifier

The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore (_). Identifiers are also case sensitive in C. For example name and Name are two different identifier in C. Keywords are not allowed to be used as Identifiers.

#475 [Explained](#) [Report](#) [Bookmark](#)

A single-line comment in C language source code can begin with _____

null

- A
- ;
- B
- :
- C
- /*
- D
- //

Correct Answer :D

Explanation

Single Line Comment Starts with '//' Any Symbols written after '//' are ignored by Compiler

#476 [Explained](#) [Report](#) [Bookmark](#)

Choose the invalid identifier from the below

null

- A Int
- B volatile
- C DOUBLE
- D None

Correct Answer :B

Explanation

volatile is a keyword in C programming language and it cannot be used as an identifier. C programming language is a case sensitive language and slight manipulation in keywords of C programming language i.e. changing from lower case to uppercase will allow us to use keywords as identifiers.

#477 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
int* f()
```

```
{
```

```
    int x = 5;
```

```
    return &x;
```

```
}
```

```
main()

{

    printf("%d", *f());
}
```

- A
5
- B
Address of 'x'
- C
Compile error
- D
Runtime error

Correct Answer :D

Explanation

Segmentation fault (core dumped) i.e. Core Dump/Segmentation fault is a specific kind of error caused by accessing memory that “does not belong to you.”

Here a location of x is returned. I.e. trying to do a read and write operation in a read only location in memory or freed block of memory, it is known as core dump.

It is an error indicating memory corruption.

#478 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    char *p = NULL;
    printf("%c", *p);
}
```

- A
NULL
- B
0
- C
Compile error
- D
Runtime error

Correct Answer :D

Explanation

In the program x points the NULL address. It is invalid to access the NULL address hence the program gives Runtime error.

Therefore the output of the program is Runtime error.

#479 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```

#include<stdio.h>

void f() {

static int i= 3;

printf("%d ", i); if(--i) f(); }main() {

f(); }

```

- A
3 2 1 0
- B
3 2 1
- C
3 3 3
- D
Compile error

Correct Answer :B

Explanation

The variable "i" is declared as static, hence memory for I will be allocated for only once, as it encounters the statement.

The function f() will be called recursively unless i becomes equal to 0, and since f() is recursively called and value i decremented by 1 , so the value of static I ie., 3, 2, 1 will be printed every time the control is returned.

What is the output of the following program?

```
#include<stdio.h>

int x = 5; int* f() {
    return &x;
}

main() {
    *f() = 10;
    printf("%d", x);
}
```

- A
Compile error
- B
Runtime error
- C
5
- D
10

Correct Answer :D

Explanation

`*f() = 10;`

`*f` function will be called to which `10` value will be passed as an argument.

```
return &x;
```

Accepted `10` values will be assigned to variable `x`. Value stored at `x` will be passed to the called environment.

```
printf("%d", x);
```

`x` printed i.e. `10`

#481 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the below code snippet.

```
#include<stdio.h>

main() {
    printf("%d", -11%2);
}
```

- A
1
- B
-1
- C
5.5
- D
-5.5

Correct Answer :B

Explanation

The % operator in C is the “modulo” operator, which has the same precedence as the * (multiply) and / (divide) operators. Its value is the remainder of the integer division of its two operands.

#482 [Explained](#) [Report](#) [Bookmark](#)

- For the below definition what is the data type of ‘PI’ #define PI 3.141

- **A**
Its float
- **B**
Its double
- **C**
There is no type associated with PI, as it's just a text substitution
- **D**
Syntax error, semi colon is missing with the definition of PI

Correct Answer :C

Explanation

PI has no data type associated with it but a text which will be placed in place of PI when program will be preprocessed.

#483 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    int a[] = {10, 20, 30};
```

```
printf("%d", *a+1); }
```

- A 10
- B 20
- C 11
- D 21

Correct Answer :C

Explanation

*a+1 will be written as ((*a)+1) and *a will be evaluated first and gives 10 as the output. In 10, 1 will be added later and output will be turned out to be 11.

#484 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
void f(int a[]) {
```

```
    int i;
```

```
    for(i=0; i<3; i++)
```

```
        a[i]++; }
```

```
main() {
```

```
int i,a[] = {10, 20, 30};

f(a); for(i=0; i<3; ++i) {

printf("%d ",a[i]); }

}
```

- A
10 20 30
- B
11 21 31
- C
Compile error
- D
Runtime error

Correct Answer :B

Explanation

```
#include<stdio.h>

void f(int a[])
{
    int i;
    for(i=0; i<3; i++)
        a[i]++;
}
```

```
// Incremented the array index value as its passes

//the reference o original value is Incremented by 1

}

main() {

    int i,a[] = {10, 20, 30};

    f(a); // Passing Array reference to function

    for(i=0; i<3; ++i)

    {

        printf("%d ",a[i]); // Printing the updated Array : 11
21 31

    }

}
```

#485 [Explained](#) [Report](#) [Bookmark](#)

Which of the following is used in mode string to open the file in binary mode?

- A
b
- B
B
- C
bin
- D
a

Correct Answer :A

Explanation

The `open()` function opens a file in text format by default. To open a file in binary format, add 'b' to the mode parameter

#486 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    char s[] = "Fine"; *s = 'N';
    printf("%s", s); }
```

- A
Fine
- B
Nine
- C
Compile error

- D
Runtime error

Correct Answer :B

Explanation

s is the array name which indicates the first element of the array.

*s helps to access first element of the array.

first character of s string f will be replaced with n so Nine will be printed

#487 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    char *s = "Fine"; *s = 'N';
    printf("%s", s); }
```

- A
Fine
- B
Nine
- C
Compile error

- D
Runtime error

Correct Answer :D

Explanation

here firstly warning will arriving for the default return type accepted will be int.

Runtime error. *s='N', trying to change the character at base address to 'N' of a constant string leads to runtime error.

#488 [Explained](#) [Report](#) [Bookmark](#)

What is the built in library function to compare two strings?

- A
string_cmp()
- B
strcmp()
- C
- equals()
- D
str_compare()

Correct Answer :B

Explanation

strcmp() is the built in library function to compare two strings.

#489 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    char s1[50], s2[50] = "Hello";
    s1 = s2; printf("%s", s1); }
```

- A Hello
- B No output
- C Compile error
- D Runtime error

Correct Answer :C

Explanation

Compiler error will be like this:

[Error] assignment to expression with array type.

You cannot assign to an array type. Array types are not suitable for a LHS argument for an assignment operator. This is what basically throws the error you see,

What is the output of the following program?

```
#include<stdio.h>

int main(); void main() {

printf("Okay");

}
```

- A No output
- B Compile error
- C segment fault error
- D okay

Correct Answer :B

Explanation

you cannot have more than one main() function in C language. In standard C language, the main() function is a special function that is defined as the entry point of the program. There cannot be more than one copy of ANY function you create in C language,

#491 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

void main() {
```

```
char s[] = "C++";  
  
printf("%s ",s); s++; printf("%s",s); }
```

- A C++ C++
- B C++ ++
- C ++ ++
- D Error

Correct Answer :D

Explanation

Array name cannot be modified i.e. it cannot be manipulated to point another element except first element. Compiler will be like this:

[Error] lvalue required as increment operand.

#492 [Explained](#) [Report](#) [Bookmark](#)

A local variable is stored in ____

- A Code segment
- B Heap segment
- C Stack segment
- D None of the above

Correct Answer :C

Explanation

All allocation made by malloc(), calloc() or realloc() are stored on the heap, while all local variables are stored on the stack. All global and static variables are stored in the data segment, while constants are stored in the code segment

#493 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following statement?

```
#include<stdio.h>

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

- A
2
- B
-2
- C
1
- D
-1

Correct Answer :B

Explanation

The left-shift operator causes the bits in shift-expression to be shifted to the left by the number of positions specified by additive-expression. The bit positions that have been vacated by the shift operation are zero-filled.

#494 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    int x = 3;
    x += 2; x += 2; printf("%d", x); }
```

- A
2
- B
5
- C
7
- D
Error

Correct Answer :A

Explanation

when $x =+2$ that means you assign a positive number to x

so x will print 2 not 5

What is the output of the following program?

```
#include<stdio.h>

main() {
    char*s = "Abc";
    while(*s)
        printf("%c", *s++);
}
```

- **A**
Abc
- **B**
bc
- **C**
Compile error
- **D**
Runtime error

Correct Answer :A

Explanation

while loop will be terminated when null is encountered which will be present at the end of the string.Every time nest address value will printed

#496 Explained Report Bookmark

What is the output of the following program?

```
#include<stdio.h>

main() {
    char s[20] = "Hello\0Hi";
    printf("%d %d", strlen(s), sizeof(s));
}
```

- A
5 9
- B
7 20
- C
5 20
- D
8 20

Correct Answer :C

Explanation

`strlen(s)` will give 5 because it will count letters till null is encountered. The size of `s` is 20.

#497 Explained Report Bookmark

What is the output of the following program?

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

```
main() {  
  
char s[] = "Hello\0Hi";  
  
printf("%d %d", strlen(s), sizeof(s)); }
```

- A
5 9
- B
7 20
- C
5 20
- D
8 20

Correct Answer :A

Explanation

`strlen(s)` will give 5 because it will count letters till null is encountered. The size of s will be 9.

#498 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following statement?

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

```
main() {  
  
printf("%d", !0<2); }
```

- A
0

- **B**
1
- **C**
False
- **D**
True

Correct Answer :B

Explanation

`!0` will be evaluated first due to higher precedence of `!` operator than `<` operator and output of `!0` will be 1. Now `1<2` will be evaluated

#499 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

#define sqr(i) i*i

main() {

printf("%d %d", sqr(3), sqr(3+1));

}
```

- **A**
9 16
- **B**
9 7
- **C**
Error: macro cannot be defined in lower case.

- D
None of the above.

Correct Answer :B

Explanation

sqr(3) will be replaced by the expression $3*3$ which will turn out as 9 and sqr(3+1) will be replaced by the expression $3+1*3+1$ which will turn out as 7.

#500 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

main() {
    char*s = "Hello";
    while(*s!=NULL) printf("%c", *s++);
}
```

- A
Hello
- B
Helloelloollooo
- C
ello
- D
Compile error

Correct Answer :A

Explanation

while loop will be executed until *s==NULL is encountered

#501 [Explained](#) [Report](#) [Bookmark](#)

Which of the following functions disconnects the stream from the file pointer.

- A
fremove()
- B
remove()
- C
fclose()
- D
file pointer to be set to NULL

Correct Answer :C

Explanation

fclose() function disconnects the stream from the file pointer.

#502 [Explained](#) [Report](#) [Bookmark](#)

How do you specify double constant 3.14 as a long double?

null

- A
By using LD after 3.14

- **B**
By using L after 3.14
- **C**
By using DL after 3.14
- **D**
By using LF after 3.14

Correct Answer :B

Explanation

Double constant 3.14 as a long double will be 3.14L.

#503 [Explained](#) [Report](#) [Bookmark](#)

In the standard library of C programming language, which of the following header file is designed for basic mathematical operations?

null

- **A**
`math.h`
- **B**
`conio.h`
- **C**
`dos.h`
- **D**
`stdio.h`

Correct Answer :A

Explanation

math.h is a header file in the standard library of the C programming language designed for basic mathematical operations. Most of the functions involve the use of floating point numbers.

#504 [Explained](#) [Report](#) [Bookmark](#)

Where to place “f” with a double constant 3.14 to specify it as a float?

null

- A
(float)(3.14)(f)
- B
(f)(3.14)
- C
3.14f
- D
f(3.14)

Correct Answer :C

Explanation

To specify a double constant 3.14 as a float then f should be put after 3.14 like this 3.14f.

#505 [Explained](#) [Report](#) [Bookmark](#)

Which header file supports the functions - malloc() and calloc()?

null

- A
stdlib.h
- B
memory.h

- **C**
math.h
- **D**
stdio.h

Correct Answer :A

Explanation

To use the malloc() &calloc() functions, the standard header file to include is (naturally) “ stdlib. h ”.

#506 [Explained](#) [Report](#) [Bookmark](#)

Which of the following operator can be used to access value at address stored in a pointer variable?

null

- **A**
*
- **B**
#
- **C**
&&
- **D**
@

Correct Answer :A

Explanation

The dereference operator or indirection operator, sometimes denoted by " * " (i.e. an asterisk), is a unary operator (i.e. one with a single

operand) found in C-like languages that include pointer variables. It operates on a pointer variable, and returns an l-value equivalent to the value at the pointer address.

#507 [Explained](#) [Report](#) [Bookmark](#)

What function can be used to free the memory allocated by calloc()?

null

- A
 dealloc();
- B
 strcat();
- C
 free();
- D
 memcpy();

Correct Answer :C

Explanation

Free function is used to deallocate memory which is allocated using malloc, calloc, etc. delete operator is used to deallocate memory, which is allocated using new operator.

#508 [Explained](#) [Report](#) [Bookmark](#)

A bitwise operator “&” can turn-off a particular bit into a number.

null

- A
 Yes

- **B**
 &&
- **C**
 *
- **D**
 ||

Correct Answer :A

Explanation

To turn off the particular bit in given number we use bitwise '<<', '&' and '~' operators.

#509 [Explained](#) [Report](#) [Bookmark](#)

Which header file can be used to define input/output function prototypes and macros?

null

- **A**
 math.h
- **B**
 memory.h
- **C**
 stdio.h
- **D**
 dos.h

Correct Answer :C

Explanation

Stdio. h, which stands for "standard input/output header", is the header in the C standard library that contains macro definitions, constants, and declarations of functions and types used for various standard input and output operations.

#510 [Explained](#) [Report](#) [Bookmark](#)

Which library functions help users to dynamically allocate memory?

null

- A memalloc()and alloc()
- B malloc() and memalloc()
- C malloc() and calloc()
- D memalloc() and calloc()

Correct Answer :C

Explanation

To allocate memory dynamically, library functions are malloc() , calloc() , realloc() and free() are used. These functions are defined in the <stdlib>.

#511 [Explained](#) [Report](#) [Bookmark](#)

Which standard library function can return a pointer to the last occurrence of a character in a string?

null

- A
stchar()
- B
strrchr()
- C
strchar() &stchar()
- D
strrchar()

Correct Answer :B

Explanation

strrchr() returns a pointer to the last occurrence of character in a string.

#512 [Explained](#) [Report](#) [Bookmark](#)

In the given below code, if a short int value is 5 byte long, then how many times the while loop will get executed?

```
#include<stdio.h>

int main () {

int j = 1; while(j <= 300) {

printf("%c %d\n", j, j); j++; }

return 0; }
```

- A
Unlimited times
- B
0 times

- C
300 times
- D
5 times

Correct Answer :C

Explanation

while loop will be executed 300 times until condition j++ becomes 301

#513 [Explained](#) [Report](#) [Bookmark](#)

Stderr" is a standard error.

- A
Yes
- B
Standard error streams
- C
Standard error types
- D
Standard error function

Correct Answer :A

Explanation

Stderr, also known as standard error, is the default file descriptor where a process can write error messages. Its default file descriptor number is 2. In the terminal, standard error defaults to the user's screen.

#514 [Explained](#) [Report](#) [Bookmark](#)

Which library function can convert an integer/long to a string?

null

- A `Itoa()`
- B `ultoa()`
- C `sprintf()`
- D **None of the above**

Correct Answer :A

Explanation

Function Name	Purpose
---------------	---------

<code>itoa()</code>	Converts an integer value to a string.
---------------------	--

Itoa() Converts a long integer value to a string.

ultoa() Converts an unsigned long integer value to a string.

#515 Explained Report Bookmark

Which library function can convert an unsigned long to a string?

null

- A
Itoa()
- B
ultoa()

- **C**
`system()`
- **D**
`unsigned long can't be converted to a string`

Correct Answer :B

Explanation

`ultoa` **Conve**
() **rts an**
 unsign
 ed
 long
 integer
 value
 to a
 string.

#516 [Explained](#) [Report](#) [Bookmark](#)

Why to use `fflush()` library function?

`null`

- **A**
To flush all streams and specified streams
- **B**
To flush only specified stream
- **C**
To flush input/output buffer
- **D**
Invalid library function

Correct Answer :A

Explanation

`fflush()` is typically used for output stream only. Its purpose is to clear (or flush) the output buffer and move the buffered data to console (in case of `stdout`) or disk (in case of file output stream). C uses a buffer to output or input variables. The buffer stores the variable that is supposed to be taken in (input) or sent out (output) of the program. A buffer needs to be cleared before the next input is taken in.

#517 [Explained](#) [Report](#) [Bookmark](#)

Which of the following variable cannot be used by switch-case statement?

- **A**
`char`
- **B**
`int`
- **C**
`float`
- **D**
`None of the above`

Correct Answer :C

Explanation

A switch works with the `byte` , `short` , `char` , and `int` primitive data types. It also works with enumerated types (discussed in Enum Types), the `String` class, and a few special classes that wrap certain primitive types: `Character` , `Byte` , `Short` , and `Integer`

#518 [Explained](#) [Report](#) [Bookmark](#)

How many times the given below program will print "IndiaPIN"?

```
#include<stdio.h>

int main ()
{
    printf("IndiaPIN");

    main();
    return 0;
}
```

- A
0 times
- B
100 times
- C
Unlimited times
- D
Till stack run over

Correct Answer :D

Explanation

If a program uses more memory space than the stack size then stack overflow will occur and can result in a program crash. If function recursively call itself infinite times then the stack is unable to store large number of local variables used by every function call and will result in overflow of stack.

#519 [Explained](#) [Report](#) [Bookmark](#)

Choose the correct statement that is a combination of these two statements,

Statement 1: char *p;

Statement 2: p = (char*) malloc(100);

- A
char p = *malloc(100);
- B
char *p = (char) malloc(100);
- C
- char *p = (char*)malloc(100);
- D
None of the above

Correct Answer :B

Explanation

The below code is a prototype of malloc() function, here ptr indicates the pointer.

ptr = (data type *)malloc(size);

In below code, "*p" is a pointer of data type char and malloc() function is used for allocating the memory for char.

```
char *p = (char*)malloc(100);
```

#520 [Explained](#) [Report](#) [Bookmark](#)

In the given below code, the P2 is

```
Typedef int *ptr;
```

```
ptr p1, p2;
```

- A Integer
- B Integer pointer
- C Both, Integer & Integer pointer
- D None of above

Correct Answer :B

Explanation

p2 is an integer pointer

#521 [Explained](#) [Report](#) [Bookmark](#)

The correct order of evaluation for the expression “z = x + y * z / 4 % 2 – 1”

- A * / % = + -
- B / * % - + =
- C - + = * % /

- D
* / % + - =

Correct Answer :D

Explanation

C uses left associativity for evaluating expressions to break a tie between two operators having same precedence

#522 [Explained](#) [Report](#) [Bookmark](#)

In C, what is the correct hierarchy of arithmetic operations?

null

- A
*/ + -
- B
* +- /
- C
/*+ -
- D
+ - / *

Correct Answer :C

Explanation

Simply called as BODMAS (Bracket of Division, Multiplication, Addition and Subtraction).

How Do I Remember ? BODMAS !

- B - Brackets first
- O - Orders (ie Powers and Square Roots, etc.)
- DM - Division and Multiplication (left-to-right)
- AS - Addition and Subtraction (left-to-right)

#523 Explained Report Bookmark

Which of the following is the correct usage of conditional operators used in C?

null

- A
`a>b ? c=30 : c=40;`
- B
`a>b ? c=30;`
- C
`max = a>b ? a>c?a:c:b>c?b:c`
- D
`return (a>b)?(a:b)`

Correct Answer :C

Explanation

Option A: assignment statements are always return in parenthesis in the case of conditional operator. It should be `a>b? (c=30):(c=40);`

Option B: it is syntactically wrong.

Option D: syntactically wrong, it should be `return(a>b ? a:b);`

Option C: it uses nested conditional operator, this is logic for finding greatest number out of three numbers.

#524 [Explained](#) [Report](#) [Bookmark](#)

Which of the following are unary operators in C?

- 1) !
- 2) sizeof
- 3) ~
- 4) &&

- A
1,2
- B
1,3
- C
2,4
- D
1,2,3

Correct Answer :D

Explanation

An operation with only one operand is called unary operation.

Unary operators:

! Logical NOT operator.

~ bitwise NOT operator.

sizeof Size-of operator.

Unary Operator	Example	Output
!	<code>int a=0;</code> <code>printf("%d",! a);</code>	1
Sizeof	<code>sizeof(char)</code>	1 (in byte)
~	<code>printf("%d",~ 3);</code>	-4

&& Logical AND is a logical operator.

Therefore, 1, 2, 3 are unary operators

#525 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

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

```
int main()
```

```

{
    int i=-3, j=2, k=0, m;

    m = ++i && ++j && ++k;

    printf("%d, %d, %d, %d\n", i, j, k, m);

    return 0;
}

```

- **A**
-2, 3, 1, 1
- **B**
2, 3, 1, 2
- **C**
1, 2, 3, 1
- **D**
3, 3, 1, 2

Correct Answer :A

Explanation

Step 1: *int i=-3, j=2, k=0, m;* here variable *i, j, k, m* are declared as an integer type and variable *i, j, k* are initialized to -3, 2, 0 respectively.

Step 2: *m = ++i && ++j && ++k;*

becomes *m = -2 && 3 && 1;*

becomes *m = TRUE && TRUE;* Hence this statement becomes TRUE. So it returns '1'(one). Hence m=1.

Step 3: `printf("%d, %d, %d, %d\n", i, j, k, m);` In the previous step the value of i,j,k are increemented by '1'(one).

Hence the output is "-2, 3, 1, 1".

#526 [Explained](#) [Report](#) [Bookmark](#)

To print a double value which format specifier can be used?

- A
%lf
- B
%L
- C
%Lf
- D
None of the above

Correct Answer :A

Explanation

The format specifier of both float and double is “%f”. Sometimes we even use “%lf” for double, but that doesn't makes any difference. The format specifier is used during input and output. It is a way to tell the compiler what type of data is in a variable during taking input using `scanf()` or printing using `printf()`.

#527 [Explained](#) [Report](#) [Bookmark](#)

Assuming, integer is 2 byte, What will be the output of the program?

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

```
int main()

{

    printf("%x\n", -2<<2);

    return 0;

}
```

- A
ffff
- B
0
- C
fff8
- D
Error

Correct Answer :C

Explanation

The integer value 2 is represented as 00000000 00000010 in binary system.

Negative numbers are represented in 2's complement method.

1's complement of 00000000 00000010 is 11111111 11111101 (Change all 0s to 1 and 1s to 0).

2's complement of 00000000 00000010 is 11111111 11111110 (Add 1 to 1's complement to obtain the 2's complement value).

Therefore, in binary we represent -2 as: 11111111 11111110.

After left shifting it by 2 bits we obtain: 11111111 11111000, and it is equal to "fff8" in hexadecimal system.

#528 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()
{
    int i=-3, j=2, k=0, m;
    m = ++i || ++j && ++k;
    printf("%d, %d, %d, %d\n", i, j, k, m);
```

```
    return 0;  
}
```

- A
1, 2, 1, 0
- B
2, 2, 0, 1
- C
-2, 2, 0, 0
- D
-2, 2, 0, 1

Correct Answer :D

Explanation

Step 1: *int i=-3, j=2, k=0, m;* here variable *i, j, k, m* are declared as an integer type and variable *i, j, k* are initialized to -3, 2, 0 respectively.

Step 2: *m = ++i // ++j && ++k;* here (++j && ++k;) this code will not get executed because ++i has non-zero value.

becomes *m = -2 // ++j && ++k;*

becomes *m = TRUE // ++j && ++k;* Hence this statement becomes TRUE. So it returns '1'(one). Hence m=1.

Step 3: *printf("%d, %d, %d, %d\n", i, j, k, m);* In the previous step the value of variable 'i' only incremented by '1'(one). The variable *j,k* are not incremented.

Hence the output is "-2, 2, 0, 1".

#529 Explained Report Bookmark

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    int x=12, y=7, z;

    z = x!=4 || y == 2;

    printf("z=%d\n", z);

    return 0;
}
```

- A
z=0
- B
z=1
- C
z=4
- D
z=2

Correct Answer :B

Explanation

Step 1: *int x=12, y=7, z;* here variable **x**, **y** and **z** are declared as an integer and variable **x** and **y** are initialized to 12, 7 respectively.

Step 2: **z = x!=4 // y == 2;**

becomes z = 12!=4 // 7 == 2;

then z = (condition true) // (condition false); Hence it returns 1. So the value of z=1.

Step 3: *printf("z=%d\n", z);* Hence the output of the program is "z=1".

#530 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    static int a[20];

    int i = 0;

    a[i] = i ;

    printf("%d, %d, %d\n", a[0], a[1], i);

    return 0;
}
```

- A
1, 0, 1
- B
1, 1, 1
- C
0, 0, 0
- D
0, 1, 0

Correct Answer :C

Explanation

Step 1: `static int a[20];` here variable `a` is declared as an integer type and `static`. If a variable is declared as `static` and it will be automatically initialized to value '0'(zero).

Step 2: `int i = 0;` here variable `i` is declared as an integer type and initialized to '0'(zero).

Step 3: `a[i] = i ; becomes a[0] = 0;`

Step 4: `printf("%d, %d, %d\n", a[0], a[1], i);`

Here `a[0] = 0`, `a[1] = 0`(because all static variables are initialized to '0') and `i = 0`.

Step 4: Hence the output is "0, 0, 0".

#531 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```

#include<stdio.h>

int main()

{
    int i=4, j=-1, k=0, w, x, y, z;

    w = i || j || k;

    x = i && j && k;

    y = i || j && k;

    z = i && j || k;

    printf("%d, %d, %d, %d\n", w, x, y, z);

    return 0;
}

```

- **A**
1, 1, 1, 1
- **B**
1, 1, 0, 1
- **C**
1, 0, 0, 1
- **D**
1, 0, 1, 1

Correct Answer :D

Explanation

Step 1: *int i=4, j=-1, k=0, w, x, y, z;* here variable *i, j, k, w, x, y, z* are declared as an integer type and the variable *i, j, k* are initialized to 4, -1, 0 respectively.

Step 2: *w = i || j || k;* becomes *w = 4 || -1 || 0;*. Hence it returns TRUE.
So, *w=1*

Step 3: *x = i && j && k;* becomes *x = 4 && -1 && 0;* Hence it returns FALSE. So, *x=0*

Step 4: *y = i || j && k;* becomes *y = 4 || -1 && 0;* Hence it returns TRUE.
So, *y=1*

Step 5: *z = i && j || k;* becomes *z = 4 && -1 || 0;* Hence it returns TRUE.
So, *z=1*.

Step 6: *printf("%d, %d, %d, %d\n", w, x, y, z);* Hence the output is "1, 0, 1, 1".

#532 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    int i=-3, j=2, k=0, m;
    m = ++i && ++j || ++k;
```

```
    printf("%d, %d, %d, %d\n", i, j, k, m);  
  
    return 0;  
  
}
```

- A
1, 2, 0, 1
- B
-3, 2, 0, 1
- C
-2, 3, 0, 1
- D
2, 3, 1, 1

Correct Answer :C

Explanation

Step 1: *int i=-3, j=2, k=0, m;* here variable *i, j, k, m* are declared as an integer type and variable *i, j, k* are initialized to -3, 2, 0 respectively.

Step 2: *m = ++i && ++j // ++k;*

becomes *m = (-2 && 3) // ++k;*

becomes *m = TRUE // ++k;*

(++k) is not executed because *(-2 && 3)* alone return TRUE.

Hence this statement becomes TRUE. So it returns '1'(one). Hence *m=1*.

Step 3: `printf("%d, %d, %d, %d\n", i, j, k, m);` In the previous step the value of i,j are increemented by '1'(one).

Hence the output is "-2, 3, 0, 1".

#533 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    int x=4, y, z;

    y = --x;

    z = x--;

    printf("%d, %d, %d\n", x, y, z);

    return 0;
}
```

- A
4, 3, 3
- B
4, 3, 2
- C
3, 3, 2
- D
2, 3, 3

Correct Answer :D

Explanation

Step 1: *int x=4, y, z;* here variable x, y, z are declared as an integer type and variable x is initialized to 4.

Step 2: *y = --x;* becomes *y = 3;* because *(--x)* is pre-decrement operator.

Step 3: *z = x--;* becomes *z = 3;*. In the next step variable x becomes 2, because *(x--)* is post-decrement operator.

Step 4: *printf("%d, %d, %d\n", x, y, z);* Hence it prints "2, 3, 3".

#534 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    int a=100, b=200, c;

    c = (a == 100 || b > 200);

    printf("c=%d\n", c);

    return 0;
}
```

}

- A
c=100
- B
c=200
- C
c=1
- D
c=300

Correct Answer :C

Explanation

Step 1: *int a=100, b=200, c;*

Step 2: *c = (a == 100 || b > 200);*

becomes *c = (100 == 100 || 200 > 200);*

becomes *c = (TRUE || FALSE);*

becomes *c = (TRUE);(ie. c = 1)*

Step 3: *printf("c=%d\n", c);* It prints the value of variable *i=1*

Hence the output of the program is '1'(one).

#535 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{

    int x=55;

    printf("%d, %d, %d\n", x<=55, x=40, x>=10);

    return 0;

}
```

- **A**
1, 40, 1
- **B**
1, 55, 1
- **C**
1, 55, 0
- **D**
1, 1, 1

Correct Answer :A

Explanation

Step 1: *int x=55;* here variable x is declared as an integer type and initialized to '55'.

Step 2: *printf("%d, %d, %d\n", x<=55, x=40, x>=10);*

In printf the execution of expressions is from Right to Left.

here $x \geq 10$ returns TRUE hence it prints '1'.

$x = 40$ here x is assigned to 40 Hence it prints '40'.

$x \leq 55$ returns TRUE. hence it prints '1'.

Step 3: Hence the output is "1, 40, 1"

#536 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()
{
    int k, num=30;

    k = (num>5 ? (num <=10 ? 100 : 200) : 500);

    printf("%d\n", num);

    return 0;
}
```

- A
200

- **B**
30
- **C**
100
- **D**
500

Correct Answer :B

Explanation

Step 1: *int k, num=30;* here variable **k** and **num** are declared as an integer type and variable **num** is initialized to '30'.

Step 2: *k = (num>5 ? (num <=10 ? 100 : 200): 500);* This statement does not affect the output of the program. Because we are going to print the variable **num** in the next statement. So, we skip this statement.

Step 3: *printf("%d\n", num);* It prints the value of variable **num** '30'

Step 3: Hence the output of the program is '30'

#537 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

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

```
int main()
```

```
{
```

```

char ch;

ch = 'A';

printf("The letter is");

printf("%c", ch >= 'A' && ch <= 'Z' ? ch + 'a' - 'A':ch);

printf("Now the letter is");

printf("%c\n", ch >= 'A' && ch <= 'Z' ? ch : ch + 'a' -
'A');

return 0;

}

```

- **A**
The letter is a
Now the letter is A
- **B**
The letter is A
Now the letter is a
- **C**
Error
- **D**
None of above

Correct Answer :A

Explanation

Step 1: *char ch;* *ch = 'A';* here variable *ch* is declared as an character type an initialized to 'A'.

Step 2: `printf("The letter is");` It prints "The letter is".

Step 3: `printf("%c", ch >= 'A' && ch <= 'Z' ? ch + 'a' - 'A':ch);`

The ASCII value of 'A' is 65 and 'a' is 97.

Here

=> ('A' >= 'A' && 'A' <= 'Z') ? (A + 'a' - 'A'):('A')

=> (TRUE && TRUE) ? (65 + 97 - 65) : ('A')

=> (TRUE) ? (97): ('A')

In `printf` the format specifier is '%c'. Hence prints 97 as 'a'.

Step 4: `printf("Now the letter is");` It prints "Now the letter is".

Step 5: `printf("%c\n", ch >= 'A' && ch <= 'Z' ? ch : ch + 'a' - 'A');`

Here => ('A' >= 'A' && 'A' <= 'Z') ? ('A') : (A + 'a' - 'A')

=> (TRUE && TRUE) ? ('A') :(65 + 97 - 65)

=> (TRUE) ? ('A') : (97)

It prints 'A'

Hence the output is

The letter is a

Now the letter is A

#538 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    int i=2;

    int j = i + (1, 2, 3, 4, 5);

    printf("%d\n", j);

    return 0;
}
```

- **A**
4
- **B**
7
- **C**
6
- **D**
5

Correct Answer :B

Explanation

Because, comma operator used in the expression $i(1, 2, 3, 4, 5)$. The comma operator has left-right associativity. The left operand is always evaluated first, and the result of evaluation is discarded before the right operand is evaluated. In this expression 5 is the right most operand, hence after evaluating expression $(1, 2, 3, 4, 5)$ the result is 5, which on adding to i results into 7.

#539 [Explained](#) [Report](#) [Bookmark](#)

In which header file is the NULL macro defined?

- **A**
stddef.h
- **B**
stdio.h
- **C**
stdio.h and stddef.h
- **D**
math.h

Correct Answer :C

Explanation

The macro "NULL" is defined in locale.h, stddef.h, stdio.h, stdlib.h, string.h, time.h, and wchar.h.

#540 [Explained](#) [Report](#) [Bookmark](#)

How many bytes are occupied by near, far and huge pointers (DOS)?

null

- **A**
near=2 far=4 huge=4

- **B**
`near=4 far=8 huge=8`
- **C**
`near=2 far=4 huge=8`
- **D**
`near=4 far=4 huge=8`

Correct Answer :A

Explanation

near=2, far=4 and huge=4 pointers exist only under DOS. Under windows and Linux every pointers is 4 bytes long.

#541 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
main()

{

    static int var = 5;

    printf("%d ",var--);

    if(var)

        main();

}
```

- A
5 4 3 2 1
- B
5 3 2 1
- C
5 4 3 1
- D
4 3 2 1 0

Correct Answer :A

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.

#542 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
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;      }

}

```

- **A**
2 2 2 2 2 2 3 4 6 5
- **B**
2 2 2 2 2 3 4 6 5
- **C**
2 2 2 2 2 3 4 6 5 4 3
- **D**
2 2 3 4 6 5 4 3 2 1

Correct Answer :A

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

#543 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
main()

{
    extern int i;

    i=20;

    printf("%d",i);

}
```

- **A**
20
- **B**
0
- **C**
compile time error
- **D**
run time error

Correct Answer :C

Explanation

Linker Error : Undefined symbol '_i'

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 .

#544 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
main()

{

    char *p;

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

}
```

- A
1 4
- B
1 3
- C
1 2
- D
Error

Correct Answer :C

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.

#545 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

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

- A
0 F F F
- B
F F F 0
- C
F F F 1
- D
0 0 0 0

Correct Answer :A

Explanation

Turbo C Compilation

They are asking to print ("%x",-1>>4);

"%x" means your integer will be displayed in hexadecimal value.

Lets take this A/C to Answer given write the 1 in 16 bits

0000 0000 0000 0001

-1 can be written in 2's complement no system as

1111 1111 1111 1111

>> This Operator is called Binary Right Shift Operator ,So we have to right shift the o/p bits and fill the shifted bits with 0

0000 1111 1111 1111

0 F F F

so, on a computer with 32-bit int you'll have:

```
11111111111111111111111111 = -1 (if interpreted as a  
signed integer)
```

now, if you shift this to the left of 4 positions, you get:

```
1111111111111111111111110000
```

#546 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

main()
{
    int i=10;

    i=!i>14;

    printf("i=%d",i);

}
```

- A
i=0
- B
i=10
- C
i=1
- D
None

Correct Answer :A

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).

#547 Explained Report Bookmark

What will be the output of the program?

```
#include<stdio.h>

#define clrscr() 100

main()

{

clrscr();

printf("%d\n",clrscr());
}
```

- A
Compile Time Error
- B
Runtime Error
- C
100
- D
No output

Correct Answer :C

Explanation

Preprocessor executes as a separate pass before the execution of the compiler. So textual replacement of `clrscr()` to 100 occurs. The input program to compiler looks like this :

```
main(){

100;

printf("%d\n",100);

}
```

Note: 100; is an executable statement but with no action. So it doesn't give any problem

#548 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program? 10 is given as input here

```
main()

{

int i;

printf("%d",scanf("%d",&i));

}
```

- A
10
- B
0
- C
1
- D
No Output

Correct Answer :C

Explanation

Scnf returns number of items successfully read and not 1/0. Here 10 is given as input which should have been scanned successfully. So number of items read is 1.

#549 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    unsigned int i = 65536; /* Assume 2 byte integer*/

    while(i != 0)

        printf("%d",++i);

    printf("\n");

    return 0;
}
```

- A
Infinite loop
- B
0 1 2 ... 65535

- **C**
0 1 2 ... 32767 - 32766 -32765 -1 0
- **D**
No output

Correct Answer :D

Explanation

Here unsigned int size is 2 bytes. It varies from 0,1,2,3, ... to 65535.

Step 1: unsigned int i = 65536; here variable i becomes '0'(zero). because unsigned

int varies from 0 to 65535.

Step 2: while(i != 0) this statement becomes while(0 != 0). Hence the while(FALSE)

condition is not satisfied. So, the inside the statements of while loop will not get executed.

Hence there is no output.

Note: Don't forget that the size of int should be 2 bytes. If you run the above program in

GCC it may run infinite loop, because in Linux platform the size of the integer is 4 bytes.

Which of the following correctly represents a long double constant?

null

- A 6.68
- B 6.68L
- C 6.68f
- D 6.68LF

Correct Answer :B

Explanation

6.68 is double.

6.68L is long double constant.

6.68f is float constant.

6.68LF is not allowed in c.

#551 [Explained](#) [Report](#) [Bookmark](#)

Which of the following operations is INCORRECT?

null

- A int i = 35; i = i%5;
- B short int j = 255; j = j;

- **C**
`long int k = 365L; k = k;`
- **D**
`float a = 3.14; a = a%3;`

Correct Answer :D

Explanation

`float a = 3.14; a = a%3;` gives "Illegal use of floating point" error.

The modulus (%) operator can only be used on integer types. We have to

use `fmod()` function in `math.h` for float values.

#552 [Explained](#) [Report](#) [Bookmark](#)

Which of the declaration is correct?

null

- **A**
`int length;`
- **B**
`char int;`
- **C**
`int long;`
- **D**
`float double;`

Correct Answer :A

Explanation

int length; denotes that variable length is int(integer) data type.

char int; here int is a keyword cannot be used a variable name.

int long; here long is a keyword cannot be used a variable name.

float double; here double is a keyword cannot be used a variable name.

So, the answer is int length;(Option A).

#553 [Explained](#) [Report](#) [Bookmark](#)

Point out the error in the following program.

```
#include<stdio.h>

int main()

{

    int (*p) () = fun:

        (*p) () ;

    return 0;

}

int fun ()
```

```

    printf("Placementadda\n") ;

    return 0;

}

```

- A Error: in int(*p)() = fun;
- B Error: fun() prototype not defined
- C No error
- D None of these

Correct Answer :B

Explanation

The compiler will not know that the function int fun() exists. So we have to define the function prototype of int fun(); To overcome this error, see the below program

```

#include<stdio.h>

int fun(); /* function prototype */

int main()

{
    int(*p) () = fun;

    (*p) ();
}

```

```
    return 0;

}

int fun()

{

    printf("Placementadda\n");

    return 0;

}
```

#554 Explained Report Bookmark

Which of the following is correct about err used in the declaration given below?
typedef enum error { warning, test, exception } err;

- A It is a **typedef** for enum error.
- B It is a variable of type enum error.
- C The statement is erroneous.
- D It is a structure.

Correct Answer :A

Explanation

A **typedef** gives a new name to an existing data type.

So err is a new name for enum error.

#555 [Explained](#) [Report](#) [Bookmark](#)

Point out the error in the following program.

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

```
struct emp
```

```
{
```

```
    char name[20] ;
```

```
    int age;
```

```
} ;
```

```
int main()
```

```
{
```

```
    emp int xx;
```

```
    int a;
```

```
    printf("%d\n", &a) ;
```

```
    return 0;  
}  
  
• A  
  Error: in printf  
• B  
  Error: in emp int xx;  
• C  
  No error.  
• D  
  None of these
```

Correct Answer :B

Explanation

There is an error in the line emp int xx; To overcome this error, remove the int and add the struct at the begining of emp int xx;

```
include<stdio.h>  
  
struct emp  
  
{  
  
    char name[20];  
  
    int age;  
  
};  
  
int main()
```

```
{  
  
    struct emp xx;  
  
    int a;  
  
    printf("%d\n", &a);  
  
    return 0;  
  
}
```

#556Explained Report Bookmark

What will be the output of the program?

```
#include<stdio.h>  
  
int main()  
  
{  
  
    int x = 40;  
  
    {  
  
        int x = 20;  
  
        printf("%d", x);  
  
    }  
  
    printf("%d\n", x);
```

```
    return 0;  
}
```

- A
40 40
- B
20 40
- C
20
- D
Error

Correct Answer :B

Explanation

In case of a conflict between a local variable and global variable,
the local variable gets priority.

#557 [Explained](#) [Report](#) [Bookmark](#)

In the following program how long will the for loop get executed?

```
#include  
  
int main()  
  
{  
  
    int i = 5;  
  
    for(;scanf("%s", &i); printf("%d\n", i));
```

```
    return = 0;  
}  
  
• A  
  The for loop would not get executed at all  
• B  
  The for loop would get executed only once  
• C  
  The for loop would get executed 5 times  
• D  
  The for loop would get executed infinite times
```

Correct Answer :D

Explanation

During the for loop execution scanf() ask input and then printf() prints that given input.

This process will be continued repeatedly because, scanf() returns the number of input

given, the condition is always true(user gives a input means it returns '1').

Hence this for loop would get executed infinite times

#558 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program

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

```

int main()

{
    int a[5] = {2,3};

    printf("%d, %d, %d\n", a[2], a[3], a[4]);

    return = 0;

}

```

- A Garbage Values
- B 2, 3, 3
- C 3, 2, 2
- D 0, 0, 0

Correct Answer :D

Explanation

When an automatic array is partially initialized, the remaining elements are initialized to 0.

#559 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program

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

```
int main()

{

    extern int fun (float);

    int a;

    a = fun(3.14);

    printf("%d\n", a);

    return 0;
}
```

```
int fun (int aa)

{

    return (int)++aa;
}
```

- A
3
- B
3.14
- C
0
- D
Error

Correct Answer :D

Explanation

2 Errors

1. Type mismatch in redeclaration of fun

2. Type mismatch in parameter aa

#560 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program

```
#include<stdio.h>

int main()
{
    int x = 10, y = 20, z = 5, i ;
    i = x < y < z;
    printf ("%d\n", i);
    return 0;
}
```

- A
0
- B
1
- C
Error

- D
- None of these**

Correct Answer :B

Explanation

Since $x < y$ turns to be TRUE it is replaced by 1. Then $1 < z$ is compared and to be TRUE.

The 1 is assigned to i.

#561 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program

```
#include<stdio.h>

int main ()
{
    struct emp
    {
        char name [20] ;
        int age;
        float sal;
    };
}
```

```

struct emp e = {"Tiger"};  
  

printf ("%d, %f\n", e.age, e.sal);  
  

return 0;  
}

```

- **A**
0, 0.000000
- **B**
Garbage values
- **C**
Error
- **D**
None of above

Correct Answer :A

Explanation

When an automatic structure is partially initialized remaining elements are

initialized to 0(zero).

#562 [Explained](#) [Report](#) [Bookmark](#)

By default a real number is treated as a

- **A**
float

- **B**
double
- **C**
long double
- **D**
far double

Correct Answer :B

Explanation

In computing, 'real number' often refers to non-complex floating-point numbers. It

include both rational numbers, such as 42 and 3/4, and irrational numbers such

as pi = 3.14159265...

When the accuracy of the floating point number is insufficient, we can use the double to

define the number. The double is same as float but with longer precision and takes double

space (8 bytes) than float.

To extend the precision further we can use long double which occupies 10 bytes of memory

space.

#563 Explained Report Bookmark

What will be output when you will execute following c code?

```
#include<stdio.h>

int main()

{
    printf("%d\t", sizeof(6.5));
    printf("%d\t", sizeof(90000));
    printf("%d\t", sizeof('A'));
    return 0;
}
```

- A
4 2 1
- B
8 2 1
- C
8 4 1
- D
8 4 2

Correct Answer :D

Explanation

Output on Turbo C++ 3.0:

8 4 2

Output on Turbo C++ 4.5:

8 4 2

Output on Linux GCC:

8 4 4

Output on Visual C++:

8 4 4

By default data type of numeric constants is:

6.5 : double

90000: long int

'A': char

In C size of data type varies from compiler to compiler.

#564 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    int a = 500, b = 100, c;

    if(!a >= 400)

        b = 300;

    c = 200;

    printf("b = %d c = %d\n", b, c);

    return 0;
}
```

- **A**
b = 300 c = 200
- **B**
b = 100 c = garbage
- **C**
b = 300 c = garbage
- **D**
b = 100 c = 200

Correct Answer :D

Explanation

Initially variables a = 500, b = 100 and c is not assigned.

Step 1: if(!a >= 400)

Step 2: if(!500 >= 400)

Step 3: if(0 >= 400)

Step 4: if(FALSE) Hence the if condition is failed.

Step 5: So, variable c is assigned to a value '200'.

Step 6: printf("b = %d c = %d\n", b, c); It prints value of b and c.

Hence the output is "b = 100 c = 200"

#565 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
    char str[]="C-program";

    int a = 5;
```

```
    printf(a >10?"Ps\\ n": "%s\\ n", str);  
  
    return 0;  
  
}
```

- A C-program
- B Ps
- C Error
- D None of above

Correct Answer :A

Explanation

Step 1: char str[]="C-program"; here variable str contains "C-program".

Step 2: int a = 5; here variable a contains "5".

Step 3: printf(a >10?"Ps\\ n": "%s\\ n", str);

this statement can be written as

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

```
if(a > 10)
```

```
{
```

```
    printf("Ps\n");

}

else

{

    printf("%s\n", str);

}
```

Here we are checking `a > 10` means `5 > 10`. Hence this condition will be failed. So it prints variable `str`. Hence the output is "C-program".

#566 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{

    int i=0;

    for(; i<=5; i++);

    printf("%d", i);
```

```
    return 0;  
}
```

- A
0, 1, 2, 3, 4, 5
- B
5
- C
1, 2, 3, 4
- D
6

Correct Answer :A

Explanation

Step 1: int i = 0; here variable i is an integer type and initialized to '0'.

Step 2: for(; i<=5; i++); variable i=0 is already assigned in previous step. The semi-colon at the end of this for loop tells, "there is no more statement is inside the loop".

Loop 1: here i=0, the condition in for(; 0<=5; i++) loop satisfies and then i is incremented by '1'(one)

Loop 2: here i=1, the condition in for(; 1<=5; i++) loop satisfies and then i is incremented by '1'(one)

Loop 3: here i=2, the condition in for(; 2<=5; i++) loop satisfies and then i is incremented by '1'(one)

Loop 4: here i=3, the condition in for(; 3<=5; i++) loop satisfies and then i is incremented by '1'(one)

Loop 5: here i=4, the condition in for(; 4<=5; i++) loop satisfies and then i is incremented by '1'(one)

Loop 6: here i=5, the condition in for(; 5<=5; i++) loop satisfies and then i is incremented by '1'(one)

Loop 7: here i=6, the condition in for(; 6<=5; i++) loop fails and then i is not incremented.

Step 3: printf("%d", i); here the value of i is 6. Hence the output is '6'.

#567 [Explained](#) [Report](#) [Bookmark](#)

What would be output if you compile and execute the following C code?

```
void main()
{
    int a=5;

    begin:

    if(a)

        printf("%d    ",a);

    a--;

    goto begin;
```

}

- **A**
5 5 5 5
- **B**
5 4 3 2 1
- **C**
Infinite
- **D**
Compile Error

Correct Answer :C

Explanation

This program runs infinitely, because the goto statement is not inside the if structure.

#568 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()
{
    int x =4, y, z=2;

    y = --x;
```

```
z = x--;

printf("%d, %d, %d\n", x, y, z);

return 0;

}
```

- A
4, 3, 3
- B
4, 3, 2
- C
3, 3, 2
- D
2, 3, 3

Correct Answer :D

Explanation

Step 1: int x=4, y, z; here variable x, y, z are declared as an integer type and variable x is initialized to 4.

Step 2: y = --x; becomes y = 3; because (--x) is pre-decrement operator.

Step 3: z = x--; becomes z = 3;. In the next step variable x becomes 2, because (x--) is post-decrement operator.

Step 4: printf("%d, %d, %d\n", x, y, z); Hence it prints "2, 3, 3".

Which of the following are unary operators in C?

1. !
2. `sizeof`
3. ~
4. &&

- A
1,2
- B
2,3
- C
2,4
- D
1,2,3

Correct Answer :D

Explanation

An operation with only one operand is called unary operation.

Unary operators:

! Logical NOT operator.

~ bitwise NOT operator.

`sizeof` Size-of operator.

&& Logical AND is a logical operator.

Therefore, 1, 2, 3 are unary operators.

#570 [Explained](#) [Report](#) [Bookmark](#)

Point out the error in the program

```
#include<stdio.h>

int f(int a)

{
    a > 20? return(10): return(20);

}

int main()

{
    int f(int);
    int b;

    b = f(20);

    printf("%d\n", b);

    return 0;
}
```

- A
Error: Prototype declaration

- **B**
No error
- **C**
Error: return statement cannot be used with conditional operators
- **D**
None of above

Correct Answer :C

Explanation

In a ternary operator, we cannot use the return statement. The ternary operator

requires expressions but not code.

#571 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int fun(int i)

{
    i++;

    return i;

}
```

```
int main()
{
    int fun(int);
    int i=3;
    fun(i=fun(fun(i)));
    printf("%d\n", i);
    return 0;
}
```

- A
5
- B
4
- C
Error
- D
Garbage value

Correct Answer :A

Explanation

Step 1: int fun(int); This is prototype of function fun(). It tells the compiler that the function

fun() accept one integer parameter and returns an integer value.

Step 2: int i=3; The variable i is declared as an integer type and initialized to value 3.

Step 3: fun(i=fun(fun(i)));. The function fun(i) increments the value of i by 1(one) and return it. Lets go step by step,

=> fun(i) becomes fun(3) is called and it returns 4.

=> i = fun(fun(i)) becomes i = fun(4) is called and it returns 5 and stored in variable i.(i=5)

=> fun(i=fun(fun(i))); becomes fun(5); is called and it return 6 and nowhere the return value is stored.

Step 4: printf("%d\n", i); It prints the value of variable i.(5)

Hence the output is '5'.

#572 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

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

```
int check(int);
```

```
int main()
```

```
{
```

```
    int i=45, c;
```

```
c = check(i);

printf("%d\n", c);

return 0;

}

int check(int ch)

{

    if(ch >= 45)

        return 100;

    else

        return 10;

}
```

- A
100
- B
10
- C
1
- D
0

Correct Answer :A

Explanation

Step 1: int check(int); This prototype tells the compiler that the function check() accepts one

integer parameter and returns an integer value.

Step 2: int i=45, c; The variable i and c are declared as an integer type and i is initialized to 45.

**The function check(i) return 100 if the given value of variable i is
=>(greater than or equal to) 45,**

else it will return 10.

**Step 3: c = check(i); becomes c = check(45); The function check()
return 100 and it get stored in**

the variable c.(c = 100)

Step 4: printf("%d\n", c); It prints the value of variable c.

Hence the output of the program is '100'.

#573Explained Report Bookmark

What will be the output of the program?

```
include<stdio.h>

int addmult(int ii, int jj)

{
```

```
    int kk, ll;

    kk = ii + jj;

    ll = ii * jj;

    return (kk, ll);

}
```

```
int main()

{

    int i=3, j=4, k, l;

    k = addmult(i, j);

    l = addmult(i, j);

    printf("%d %dn", k, l);

    return 0;

}
```

- **A**
12, 12
- **B**
7, 7
- **C**
7, 12

- D
12, 7

Correct Answer :A

Explanation

Step 1: int i=3, j=4, k, l; The variables i, j, k, l are declared as an integer type and variable

i, j are initialized to 3, 4 respectively.

The function addmult(i, j); accept 2 integer parameters.

Step 2: k = addmult(i, j); becomes k = addmult(3, 4)

In the function addmult(). The variable kk, ll are declared as an integer type int kk, ll;

kk = ii + jj; becomes kk = 3 + 4 Now the kk value is '7'.

ll = ii * jj; becomes ll = 3 * 4 Now the ll value is '12'.

return (kk, ll); It returns the value of variable ll only.

The value 12 is stored in variable 'k'.

Step 3: l = addmult(i, j); becomes l = addmult(3, 4)

kk = ii + jj; becomes kk = 3 + 4 Now the kk value is '7'.

`ll = ii * jj;` becomes `ll = 3 * 4` Now the `ll` value is '12'.

`return (kk, ll);` It returns the value of variable `ll` only.

The value 12 is stored in variable 'l'.

Step 4: `printf("%d, %d\n", k, l);` It prints the value of `k` and `l`

Hence the output is "12, 12".

#574 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int func1(int);

int main()
{
    int k=35;

    k = func1(k=func1(k=func1(k)));

    printf("k=%d\n", k);

    return 0;
}
```

```
int func1(int k)

{
    k++;

    return k;
}
```

- A
k=35
- B
k=36
- C
k=37
- D
k=38

Correct Answer :D

Explanation

Step 1: int k=35; The variable k is declared as an integer type and initialized to 35.

Step 2: k = func1(k=func1(k=func1(k))); The func1(k) increments the value of k by 1

and return it. Here the func1(k) is called 3 times. Hence it increments value of k = 35

to 38. The result is stored in the variable k = 38.

Step 3: printf("k=%d\n", k); It prints the value of variable k "38".

#575 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()
{
    int fun(int);

    int i = fun(10);

    printf("%d\n", --i);

    return 0;
}

int fun(int i)
{
    return (i++);
}
```

- A
9
- B
10

- C
11
- D
8

Correct Answer :A

Explanation

Step 1: int fun(int); Here we declare the prototype of the function fun().

Step 2: int i = fun(10); The variable i is declared as an integer type and the result of the fun(10) will be stored in the variable i.

Step 3: int fun(int i){ return (i++); } Inside the fun() we are returning a value

return(i++). It returns 10. because i++ is the post-increment operator.

Step 4: Then the control back to the main function and the value 10 is assigned to variable i.

Step 5: printf("%dn", --i); Here --i denoted pre-increment. Hence it prints the value 9.

#576 [Explained](#) [Report](#) [Bookmark](#)

Which of the following are correct preprocessor directives in C?

1: #ifdef

2: #if

3:#elif

4:#undef

- A
1, 2
- B
4
- C
1, 2, 4
- D
1, 2, 3, 4

Correct Answer :D

Explanation

The macros #ifdef #if #elif are called conditional macros.

The macro #undef undefine the previosly declared macro symbol.

Hence all the given statements are macro preprocessor directives.

#577 [Explained](#) [Report](#) [Bookmark](#)

Point out the error in the program

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

```

int main()

{
    int i;

#ifndef A

    printf("Enter any number:");

    scanf("%d", &i);

#endif B

    printf("The number is odd");

    return 0;
}

```

- **A**
Error: unexpected end of file because there is no matching
#endif
- **B**
The number is odd
- **C**
Garbage values
- **D**
None of above

Correct Answer :A

Explanation

The conditional macro #if must have an #endif. In this program there is no #endif statement written.

#578 [Explained](#) [Report](#) [Bookmark](#)

Point out the error in the program

```
#include<stdio.h>

#define SI(p, n, r) float si; si=p*n*r/100;

int main()

{

    float p=2500, r=3.5;

    int n=3;

    SI(p, n, r);

    SI(1500, 2, 2.5);

    return 0;

}
```

- A
26250.00 7500.00
- B
Nothing will pri
- C
Error: Multiple declaration of si
- D
Garbage values

Correct Answer :C

Explanation

The macro **#define SI(p, n, r) float si; si=p*n*r/100;** contains the error.
To remove this error,

we have to modify this macro to

```
#define SI(p,n,r) p*n*r/100
```

#579 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

#define MAX(a, b, c) (a>b ? a>c ? a : c: b>c ? b : c)

int main()

{

    int x;

    x = MAX(3+2, 2+7, 3+7);

    printf("%d\n", x);

    return 0;
```

}

- A
5
- B
9
- C
10
- D
3+7

Correct Answer :C

Explanation

The macro **MAX(a, b, c)** ($a>b ? a>c ? a : c : b>c ? b : c$) returns the biggest of given three numbers.

Step 1: int x; The variable x is declared as an integer type.

Step 2: $x = \text{MAX}(3+2, 2+7, 3+7)$; becomes,

$\Rightarrow x = (3+2 > 2+7 ? 3+2 > 3+7 ? 3+2 : 3+7 : 2+7 > 3+7 ? 2+7 : 3+7)$

$\Rightarrow x = (5 > 9 ? (5 > 10 ? 5 : 10) : (9 > 10 ? 9 : 10))$

$\Rightarrow x = (5 > 9 ? (10) : (10))$

$\Rightarrow x = 10$

Step 3: `printf("%d\n", x);` It prints the value of 'x'.

Hence the output of the program is "10".

#580 Explained Report Bookmark

What will be the output of the program?

```
#include<stdio.h>

#define PRINT(i) printf("%d,",i)

int main()

{

    int x=2, y=3, z=4;

    PRINT(x);

    PRINT(y);

    PRINT(z);

    return 0;

}
```

- A
2, 3, 4,
- B
2, 2, 2,
- C
3, 3, 3,
- D
4, 4, 4,

Correct Answer :A

Explanation

The macro **PRINT(i)** `print("%d," , i);` prints the given variable value in an integer format.

Step 1: `int x=2, y=3, z=4;` The variable x, y, z are declared as an integer type and initialized

to 2, 3, 4 respectively.

Step 2: **PRINT(x);** becomes `printf("%d," ,x);` Hence it prints '2'.

Step 3: **PRINT(y);** becomes `printf("%d," ,y);` Hence it prints '3'.

Step 4: **PRINT(z);** becomes `printf("%d," ,z);` Hence it prints '4'.

Hence the output of the program is 2, 3, 4.

#581 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

#define str(x) #x

#define Xstr(x) str(x)

#define oper multiply
```

```
int main()

{

    char *opername = Xstr(oper) ;

    printf("%s\n", opername) ;

    return 0;

}
```

- **A**
Error: in macro substitution
- **B**
Error: invalid reference 'x' in macro
- **C**
print 'multiply'
- **D**
No output

Correct Answer :C

Explanation

The macro #define str(x) #x replaces the symbol 'str(x)' with 'x'.

The macro #define Xstr(x) str(x) replaces the symbol 'Xstr(x)' with 'str(x)'.

The macro #define oper multiply replaces the symbol 'oper' with 'multiply'.

Step 1: `char *opername = Xstr(oper);` The variable `*opername` is declared as a pointer

to a character type.

=> `Xstr(oper);` becomes,

=> `Xstr(multiply);`

=> `str(multiply)`

=> `char *opername = multiply`

Step 2: `printf("%s\n", opername);` It prints the value of variable `opername`.

Hence the output of the program is "multiply"

#582Explained Report Bookmark

What will be the output of the program?

```
#include<stdio.h>

#define MAX(a, b) (a > b ? a : b)

int main()
{
```

```

int x;

x = MAX(3+2, 2+7);

printf("%d\n", x);

return 0;

}

```

- A
8
- B
9
- C
6
- D
5

Correct Answer :B

Explanation

The macro **MAX(a, b)** ($a > b ? a : b$) returns the biggest value of the given two numbers.

Step 1 : int x; The variable x is declared as an integer type.

Step 2 : $x = \text{MAX}(3+2, 2+7);$ becomes,

$$\Rightarrow x = (3+2 > 2+7 ? 3+2 : 2+7)$$

$$\Rightarrow x = (5 > 9 ? 5 : 9)$$

=> x = 9

Step 3 : printf("%d\n", x); It prints the value of variable x.

Hence the output of the program is 9.

#583 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

#define SWAP(a, b) int t; t=a, a=b, b=t;

int main()

{

    int a=10, b=12;

    SWAP(a, b);

    printf("a = %d, b = %d\n", a, b);

    return 0;

}
```

- **A**
a = 10, b = 12
- **B**
a = 12, b = 10
- **C**
Error: Declaration not allowed in macro

- **D**
Error: Undefined symbol 't'

Correct Answer :B

Explanation

The macro SWAP(a, b) int t; t=a, a=b, b=t; swaps the value of the given two variable.

Step 1: int a=10, b=12; The variable a and b are declared as an integer type and initialized to 10, 12 respectively.

Step 2: SWAP(a, b);. Here the macro is substituted and it swaps the value to variable a and b.

Hence the output of the program is 12, 10.

#584 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

#define PRINT(int) printf("int=%d, ", int);

int main()
```

```
{  
  
    int x=2, y=3, z=4;  
  
    PRINT(x);  
  
    PRINT(y);  
  
    PRINT(z);  
  
    return 0;  
}
```

- A
int=2, int=3, int=4
- B
int=2, int=2, int=2
- C
int=3, int=3, int=3
- D
int=4, int=4, int=4

Correct Answer :A

Explanation

The macro PRINT(int) print("%d," , int); prints the given variable value in an integer format.

Step 1: int x=2, y=3, z=4; The variable x, y, z are declared as an integer type and initialized

to 2, 3, 4 respectively.

Step 2: PRINT(x); becomes printf("int=%d," ,x). Hence it prints 'int=2'.

Step 3: PRINT(y); becomes printf("int=%d," ,y). Hence it prints 'int=3'.

Step 4: PRINT(z); becomes printf("int=%d," ,z). Hence it prints 'int=4'.

Hence the output of the program is int=2, int=3, int=4.

#585 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program (turbo c compilation) ?

```
#include<stdio.h>#define CUBE (x) (x*x*x)
```

```
#define CUBE (x) (x*x*x)
```

```
int main()
```

```
{
```

```
    int a, b=3;
```

```
    a = CUBE(b++) ;
```

```
    printf("%d, %d\n", a, b) ;
```

```
    return 0;
```

```
}
```

- A
9, 4

- **B**
27, 4
- **C**
27, 6
- **D**
Error

Correct Answer :C

Explanation

The macro function CUBE(x) ($x*x*x$) calculates the cubic value of given number(Eg: 103.)

Step 1: int a, b=3; The variable a and b are declared as an integer type and variable b id

initialized to 3.

Step 2: a = CUBE(b++); becomes

=> a = b++ * b++ * b++;

=> a = 3 * 3 * 3; Here we are using post-increment operator, so the 3 is not incremented in

this statement.

=> a = 27; Here, 27 is store in the variable a. By the way, the value of variable b is incremented

by 3. (ie: b=6)

Step 3: printf("%d, %d\n", a, b); It prints the value of variable a and b.

Hence the output of the program is 27, 6.

#586Explained Report Bookmark

What will be the output of the program?

```
#include<stdio.h>

#define SQR(x) (x*x)

int main()
{
    int a, b=3;

    a = SQR(b+2);

    printf("%d\n", a);

    return 0;
}
```

- A
25
- B
11
- C
Error

- D
Garbage value

Correct Answer :B

Explanation

The macro function **SQR(x)(x*x)** calculate the square of the given number 'x'. (Eg: 10²)

Step 1: int a, b=3; Here the variable a, b are declared as an integer type and the variable b

is initialized to 3.

Step 2: a = SQR(b+2); becomes,

=> a = b+2 * b+2; Here SQR(x) is replaced by macro to x*x .

=> a = 3+2 * 3+2;

=> a = 3 + 6 + 2;

=> a = 11;

Step 3: printf("%d\n", a); It prints the value of variable 'a'.

Hence the output of the program is 11

#587 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```

#include<stdio.h>

#define SQUARE(x) x*x

int main()

{
    float s=10, u=30, t=2, a;

    a = 2*(s-u*t)/SQUARE(t);

    printf("Result = %f", a);

    return 0;
}

```

- A
Result = -100.000000
- B
Result = -25.000000
- C
Result = 0.000000
- D
Result = 100.000000

Correct Answer :A

Explanation

The macro function **SQUARE(x)** $x*x$ calculate the square of the given number 'x'. (Eg: 10²)

Step 1: float s=10, u=30, t=2, a; Here the variable s, u, t, a are declared as an floating point

type and the variable s, u, t are initialized to 10, 30, 2.

Step 2: $a = 2 * (s - u * t) / \text{SQUARE}(t)$; becomes,

$\Rightarrow a = 2 * (10 - 30 * 2) / t * t$; Here **SQUARE(t)** is replaced by macro to $t*t$.

$\Rightarrow a = 2 * (10 - 30 * 2) / 2 * 2;$

$\Rightarrow a = 2 * (10 - 60) / 2 * 2;$

$\Rightarrow a = 2 * (-50) / 2 * 2;$

$\Rightarrow a = 2 * (-25) * 2;$

$\Rightarrow a = (-50) * 2;$

$\Rightarrow a = -100;$

Step 3: `printf("Result=%f", a);` It prints the value of variable 'a'.

Hence the output of the program is -100

#588 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```

#include<stdio.h>

#define MAN(x, y) ((x)>(y)) ? (x):(y);

int main()

{
    int i=10, j=5, k=0;

    k = MAN(++i, j++) ;

    printf("%d, %d, %d\n", i, j, k);

    return 0;
}

```

- **A**
12, 6, 12
- **B**
11, 5, 11
- **C**
11, 5, Garbage
- **D**
12, 6, Garbage

Correct Answer :A

Explanation

The macro **MAN(x, y) ((x)>(y)) ? (x):(y);** returns the biggest number of given two numbers.

Step 1: int i=10, j=5, k=0; The variable i, j, k are declared as an integer type and initialized to

value 10, 5, 0 respectively.

Step 2: k = MAN(++i, j++); becomes,

=> k = ((++i)>(j++)) ? (++i):(j++);

=> k = ((11)>(5)) ? (12):(6);

=> k = 12

Step 3: printf("%d, %d, %d\n", i, j, k); It prints the variable i, j, k.

In the above macro step 2 the variable i value is incremented by 2 and variable j value is

incremented by 1.

Hence the output of the program is 12, 6, 12

#589 [Explained](#) [Report](#) [Bookmark](#)

What would be output of the program ?

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

```
#define x 5+2
```

```
void main()  
  
{  
  
    int i;  
  
    i = x*x*x;  
  
    printf("%d",i);  
  
}
```

- A
343
- B
27
- C
133
- D
Compiler Error

Correct Answer :B

Explanation

As we know #define is a token pasting preprocessor it only paste the value of micro constant in the program, before the actual compilation start.

So pasting 5+2 in place of X, we have

$i = 5 + 2 * 5 + 2 * 5 + 2$

$= 5 + 10 + 10 + 2$

$= 27$

#590 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about an array?

1: The array int num[26]; can store 26 elements.

2: The expression num[1] designates the very first element in the array.

3: It is necessary to initialize the array at the time of declaration.

4: The declaration num[SIZE] is allowed if SIZE is a macro.

- A
1
- B
1,4
- C
2,3
- D
2,4

Correct Answer :B

Explanation

1. The array int num[26]; can store 26 elements. This statement is true.

2. The expression num[1] designates the very first element in the array. This statement is

false, because it designates the second element of the array.

3. It is necessary to initialize the array at the time of declaration. This statement is false.

4. The declaration num[SIZE] is allowed if SIZE is a macro. This statement is true, because

the MACRO just replaces the symbol SIZE with given value.

Hence the statements '1' and '4' are correct statements.

#591 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about the program below?

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

```
int main()
```

```
{
```

```
    int size, i;
```

```
    scanf("%d", &size);
```

```

int arr[size];

for(i=1; i<=size; i++)

{

    scanf("%d", arr[i]);

    printf("%d", arr[i]);

}

return 0;

}

```

- **A**
The code is erroneous since the subscript for array used in for loop is in the range 1 to size.
- **B**
The code is erroneous since the values of array are getting scanned through the loop.
- **C**
The code is erroneous since the statement declaring array is invalid.
- **D**
The code is correct and runs successfully.

Correct Answer :C

Explanation

The statement `int arr[size];` produces an error, because we cannot initialize the

size of array dynamically. Constant expression is required here.

Example: int arr[10];

One more point is there, that is, usually declaration is not allowed after calling

any function in a current block of code. In the given program the declaration

int arr[10]; is placed after a function call scanf().

#592Explained Report Bookmark

What will be the output of the program if the array begins 1200 in memory?

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

```
int main()

{
    int arr[]={2, 3, 4, 1, 6};

    printf("%u, %u, %u\n", arr, &arr[0], &arr);

    return 0;
}
```

- A
1200, 1202, 1204

- **B**
1200, 1200, 1200
- **C**
1200, 1204, 1208
- **D**
1200, 1202, 1200

Correct Answer :B

Explanation

Step 1: int arr[]={2, 3, 4, 1, 6}; The variable arr is declared as an integer array and initialized.

Step 2: printf("%u, %u, %un", arr, &arr[0], &arr); Here,

The base address of the array is 1200.

=> arr, &arr is pointing to the base address of the array arr.

=> &arr[0] is pointing to the address of the first element array arr. (ie. base address)

Hence the output of the program is 1200, 1200, 1200

#593 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

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

```
int main()

{
    float arr[] = {12.4, 2.3, 4.5, 6.7};

    printf("%d\n", sizeof(arr)/sizeof(arr[0]));

    return 0;
}
```

- A
5
- B
4
- C
6
- D
7

Correct Answer :B

Explanation

The **sizeof** function return the given variable. Example: **float a=10;** **sizeof(a)** is 4 bytes

Step 1: **float arr[] = {12.4, 2.3, 4.5, 6.7};** The variable arr is declared as an floating point array

and it is initialized with the values.

Step 2: **printf("%dn", sizeof(arr)/sizeof(arr[0]));**

The variable arr has 4 elements. The size of the float variable is 4 bytes.

Hence 4 elements x 4 bytes = 16 bytes

`sizeof(arr[0])` is 4 bytes

Hence $16/4$ is 4 bytes

Hence the output of the program is '4'.

#594 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

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

```
int main()
```

```
{
```

```
    int arr[1]={10};  
  
    printf("%d\n", 0[arr]);  
  
    return 0;
```

```
}
```

- A
1

- **B**
10
- **C**
0
- **D**
6

Correct Answer :B

Explanation

Step 1: int arr[1]={10}; The variable arr[1] is declared as an integer array with size '2'

and it's first element is initialized to value '10'(means arr[0]=10)

Step 2: printf("%d\n", 0[arr]); It prints the first element value of the variable arr.

Hence the output of the program is 10.

#595 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

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

```
int main()
```

```
{
```

```

int a[5] = {5, 1, 15, 20, 25};

int i, j, m;

i = ++a[1];

j = a[1]++;

m = a[i++];

printf("%d, %d, %d", i, j, m);

return 0;
}

```

- **A**
2, 1, 15
- **B**
1, 2, 5
- **C**
3, 2, 15
- **D**
2, 3, 20

Correct Answer :C

Explanation

Step 1: int a[5] = {5, 1, 15, 20, 25}; The variable arr is declared as an integer array with a size of 5 and it is initialized to

a[0] = 5, a[1] = 1, a[2] = 15, a[3] = 20, a[4] = 25 .

Step 2: int i, j, m; The variable i,j,m are declared as an integer type.

Step 3: i = ++a[1]; becomes i = ++1; Hence i = 2 and a[1] = 2

Step 4: j = a[1]++; becomes j = 2++; Hence j = 2 and a[1] = 3.

Step 5: m = a[i++]; becomes m = a[2]; Hence m = 15 and i is incremented by 1(i++ means 2++ so i=3)

Step 6: printf("%d, %d, %d", i, j, m); It prints the value of the variables i, j, m

Hence the output of the program is 3, 2, 15

#596 [Explained](#) [Report](#) [Bookmark](#)

In C, if you pass an array as an argument to a function, what actually gets passed?

- A Value of elements in array
- B First element of the array
- C Base address of the array
- D Address of the last element of array

Correct Answer :C

Explanation

The statement 'C' is correct. When we pass an array as a function argument, the base

address of the array will be passed.

#597 [Explained](#) [Report](#) [Bookmark](#)

What will be output if you will compile and execute the following c code?

```
#include<stdio.h>

int main() {

    int a=5;

    float b;

    printf("%d",sizeof(++a+b)) ;

    printf(" %d",a) ;

    return 0;

}
```

- A
2 6
- B
4 6
- C
2 5
- D
4 5

Correct Answer :D

Explanation

++a +b

=6 + Garbage floating point number

=Garbage floating point number

//From the rule of automatic type conversion

Hence sizeof operator will return 4 because size of float data type in c is 4 byte.

Value of any variable doesn't modify inside sizeof operator. Hence value of variable a will remain 5.

#598 [Explained](#) [Report](#) [Bookmark](#)

What will be output of the following c program?

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

```
int main() {
```

```
    int goto=5;
```

```
    printf("%d", goto);
```

```
    return 0;
```

```
}
```

- A
5

- **B**
runtime error
- **C**
compile time error
- **D**
None of these

Correct Answer :C

Explanation

Invalid variable name. goto is keyword in c. variable name cannot be any keyword of c language.

#599 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the C program?

```
#include<stdio.h>

int main()
{
    int class;
    int public = 5;
    int private = 10;
    int protected = 15;
    class = public + private + protected;
```

```
    printf("%d",class);

    return 0;

}
```

- A garbage value
- B compilation error
- C Runtime error
- D 30

Correct Answer :D

Explanation

C programming don't have any access specifier. Thus it is possible to use class, Public, Private and Protected as a variable name.

#600 Explained Report Bookmark

What will be the output of the program?

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

```
int main()

{
```

```
const c = -11;

const int d = 34;

printf("%d, %d\n", c, d);

return 0;

}
```

- A
Error
- B
-11, 34
- C
11, 34
- D
- None of these
-
-

Correct Answer :B

Explanation

Step 1: const c = -11; The constant variable 'c' is declared and initialized to value "-11".

Step 2: const int d = 34; The constant variable 'd' is declared as an integer and initialized to value '34'.

Step 3: printf("%d, %dn", c, d); The value of the variable 'c' and 'd' are printed.

Hence the output of the program is -11, 34

#601 [Explained](#) [Report](#) [Bookmark](#)

If the two strings are identical, then strcmp() function returns

- A 1
- B 0
- C -1
- D none

Correct Answer :B

Explanation

Declaration: `strcmp(const char *s1, const char*s2);` The strcmp return an int value that is if $s1 < s2$ returns a value < 0 if $s1 == s2$ returns 0 if $s1 > s2$ returns a value > 0

#602 [Explained](#) [Report](#) [Bookmark](#)

Suppose n and p are unsigned int variables in a C program. We wish to set p to nC_3 . If n is large, which of the following statements is most likely to set p correctly ?

- A $p = n * (n-1) * (n-2) / 6;$
- B $p = n * (n-1) / 2 * (n-2) / 3;$
- C $p = n * (n-1) / 3 * (n-2) / 2;$
- D $p = n * (n-1) * (n-2) / 6.0;$

Correct Answer :B

Explanation

As n is large, the product $n*(n-1)*(n-2)$ will go out of the range(overflow) and it will return a value different from what is expected. Therefore, option (A) and (D) are eliminated. So we consider a shorter product $n*(n-1)$. $n*(n-1)$ is always an even number. So the subexpression " $n * (n-1) / 2$ " in option B would always produce an integer, which means no precision loss in this subexpression. And when we consider " $n*(n-1)/2*(n-2)$ ", it will always give a number which is a multiple of 3. So dividing it with 3 won't have any loss.

#603 [Explained](#) [Report](#) [Bookmark](#)

Which of the following is true

- A gets() can read a string with newline characters but a normal scanf() with %s can not.
- B gets() can read a string with spaces but a normal scanf() with %s can not.
- C gets() can always replace scanf() without any additional code
- D none

Correct Answer :B

Explanation

gets() can read a string with spaces but a normal scanf() with %s can not.

#604 [Explained](#) [Report](#) [Bookmark](#)

Predict the output of the below program:

```

#include <stdio.h>

#define EVEN 0

#define ODD 1

int main()

{

    int i = 3;

    switch (i & 1)

    {

        case EVEN: printf("Even");

                     break;

        case ODD: printf("Odd");

                     break;

        default: printf("Default");

    }

    return 0;

}

```

- A Even
- B Odd
- C Default
- D Compile-time error

Correct Answer :B

Explanation

The expression `i & 1` returns 1 if the rightmost bit is set and returns 0 if the rightmost bit is not set. As all odd integers have their rightmost bit set, the control goes to the block labeled ODD.

#605 [Explained](#) [Report](#) [Bookmark](#)

What is the output of below program?

```
#include <stdio.h>

int foo(int* a, int* b)

{
    int sum = *a + *b;
    *b = *a;
    return *a = sum - *b;
}

int main()

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

```
    return 0;  
}
```

- A1 2 1 1
- B1 1 2 1
- C1 2 2 1
- D1 2 2 2

Correct Answer :A

Explanation

The control in the logical OR goes to the second expression only if the first expression results in FALSE. The function foo() is called because i++ returns 0(post-increment) after incrementing the value of i to 1. The foo() function actually swaps the values of two variables and returns the value of second parameter. So, values of variables j and k gets exchanged and OR expression evaluates to be TRUE.

#606 [Explained](#) [Report](#) [Bookmark](#)

Predict the output?

```

#include <stdio.h>

int fun(char *str1)

{
    char *str2 = str1;

    while(*++str1);

    return (str1-str2);

}

int main()

{
    char *str = "king";

    printf("%d", fun(str));

    return 0;

}

```

- A 10
- B 09
- C 08
- D Random Number

Correct Answer :B

Explanation

The function fun() basically counts number of characters in input string. Inside fun(), pointer str2 is initialized as str1. The statement while(*++str1); increments str1 till '\0' is reached. str1 is incremented by 9. Finally the difference between str2 and str1 is returned which is 9.

#607 [Explained](#) [Report](#) [Bookmark](#)

Consider the following C program segment:

```
char p[20];  
  
char *s = "string";  
  
int length = strlen(s);  
  
int i;  
  
for (i = 0; i < length; i++)  
  
    p[i] = s[length - i];  
  
printf("%s", p);
```

- A gnirts
- B gniert
- C string
- D no output is printed

Correct Answer :D

Explanation

Let us consider below line inside the for loop `p[i] = s[length - i];` For `i = 0`, `p[i]` will be `s[6 - 0]` and `s[6]` is '`\0`'. So `p[0]` becomes '`\0`'. It doesn't matter what comes in `p[1], p[2].....` as `P[0]` will not change for `i > 0`. Nothing is printed if we print a string with first character '`\0`'

#608 [Explained](#) [Report](#) [Bookmark](#)

Predict the output of following program. Assume that the numbers are stored in 2's complement form.

```
#include

int main()

{
    unsigned int x = -1;

    int y = ~0;

    if (x == y)

        printf("same");

    else

        printf("not same");

    return 0;
}
```

- A same
- B not same
- C not same same
- D same same

Correct Answer :A

Explanation

-1 and ~0 essentially have same bit pattern, hence x and y must be same. In the comparison, y is promoted to unsigned and compared against x (See this for promotion rules). The result is “same”. However, when interpreted as signed and unsigned their numerical values will differ. x is MAXUNIT and y is -1. Since we have %u for y also, the output will be MAXUNIT and MAXUNIT.

#609 [Explained](#) [Report](#) [Bookmark](#)

Assume that size of an integer is 32 bit. What is the output of following program?

```
#include<stdio.h>

struct st

{
    int x;
    static int y;
};

int main()
{
    printf("%d", sizeof(struct st));
}
```

```
    return 0;  
}
```

- A4
- B8
- CCompiler Error
- DRuntime Error

Correct Answer :C

Explanation

In C, struct and union types cannot have static members. In C++, struct types are allowed to have static members, but union cannot have static members in C++ also.

#610 [Explained](#) [Report](#) [Bookmark](#)

What is correct about the given program?

```
#include <stdio.h>  
  
int i;  
  
int main()  
{
```

```
if (i);  
  
else  
  
    printf("Else");  
  
return 0;  
  
}
```

- A if block is executed.
- B else block is executed
- C It is unpredictable as i is not initialized
- D Error: misplaced else

Correct Answer :B

Explanation

Since i is defined globally, it is initialized with default value 0. The Else block is executed as the expression within if evaluates to FALSE. Please note that the empty block is equivalent to a semi-colon(;). So the statements if (i); and if (i) {} are equivalent.

#611 [Explained](#) [Report](#) [Bookmark](#)

Output of following program?

```
#include<stdio.h>

void dynamic(int s, ...)

{
    printf("%d ", s);

}

int main()

{
    dynamic(2, 4, 6, 8);

    dynamic(3, 6, 9);

    return 0;
}
```

- A2 3
- BCompiler Error
- C4 3
- D3 2

Correct Answer :A

Explanation

In c three continuous dots is known as ellipsis which is variable number of arguments of function. The values to parameters are assigned one by one. Now the question is how to access other arguments. See this for details.

#612 [Explained](#) [Report](#) [Bookmark](#)

Assume that the size of char is 1 byte and negatives are stored in 2's complement form

```
#include  
  
int main()  
  
{  
  
    char c = 125;  
  
    c = c+10;  
  
    printf("%d", c);  
  
    return 0;  
  
}
```

- A 135
- B +INF
- C -121
- D -8

Correct Answer :C

Explanation

125 is represented as 01111101 in binary and when we add 10 i.e 1010 in binary it becomes : 10000111. Now what does this number represent? Firstly, you should know that char can store numbers only -128 to 127 since the most significant bit is kept for sign bit. Therefore 10000111 represents a negative number. To check which number it represents we find the 2's complement of it and get 01111001 which is = 121 in decimal system. Hence, the number 10000111 represents - 121.

#613 [Explained](#) [Report](#) [Bookmark](#)

Output of following program?

```
#include <stdio.h>

int main()
{
    int i = 5;

    printf("%d %d %d", i++, i++, i++);

    return 0;
}
```

- A 7 6 5
- B 5 6 7
- C 7 7 7
- D Compiler Dependent

Correct Answer :D

Explanation

When parameters are passed to a function, the value of every parameter is evaluated before being passed to the function. What is the order of evaluation of parameters - left-to-right or right-to-left? If evaluation order is left-to-right, then output should be 5 6 7 and if the evaluation order is right-to-left, then output should be 7 6 5.

Unfortunately, there is no fixed order defined by C standard. A compiler may choose to evaluate either from left-to-right. So the output is compiler dependent.

#614 [Explained](#) [Report](#) [Bookmark](#)

What's going to happen when we compile and run the following C program?

```
#include "stdio.h"

int main()
{
    int i = 1, j;
    for ( ; ; )
    {
        if (i)
            j = --i;
        if (j < 10)
            printf("IBQuiz", j++);
    }
}
```

```
        break;

    }

    return 0;
}
```

- ACompile Error
- BNo compile error but it will run into infinite loop printing IBQuiz.
- CNo compile error and it'll print IBQuiz 10 times
- DNo compile error but it'll print IBQuiz 9 times.

Correct Answer :C

Explanation

Basically, even though the for loop doesn't have any of three expressions in parenthesis, the initialization, control and increment has been done in the body of the loop. So j would be initialized to 0 via first if. This if itself would be executed only once due to i--. Next if and else blocks are being used to check the value of j and existing the loop if j becomes 10. Please note that j is getting incremented in printf even though there's no format specifier in format string. That's why IBQuiz would be printed for j=0 to j=9 i.e. a total of 10 times.

#615 [Explained](#) [Report](#) [Bookmark](#)

Determine Output :

```
void main()

{

    int i;

    char a[]="♦";

    if(sprintf("%sn", a) )

        printf("Ok here n");

    else

        printf("Forget itn");

}
```

- A Ok here
- B Forget it
- C Error
- D None of These

Correct Answer :A

Explanation

Printf will return how many characters does it print. Hence printing a null character returns 1 which makes the if statement true, thus "Ok here" is printed.

#616 [Explained](#) [Report](#) [Bookmark](#)

Determine Output :

```
void main()
{
    int a[10];

    printf("%d %d", a[-1], a[12]);
}
```

- A 0
- B Garbage value 0
- C 0 Garbage Value
- D Garbage vlaue Garbage Value

Correct Answer :D

Explanation

In c compiler does not check array with its bounds, value at the computed location is displayed.

#617 [Explained](#) [Report](#) [Bookmark](#)

An array elements are always stored in _____ memory locations.

- A Sequential
- B Random

- **C**
Sequential and Random
- **D**
None of the above

Correct Answer :A

Explanation

When we declare an array, space is reserved in the memory of the computer for the array. The elements of the array are stored in these memory locations. The important thing about arrays is that array elements are always stored in consecutive(Sequential) memory locations

#618 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statement is correct for switch controlling expression?

- **A**Only int can be used in “switch” control expression
- **B**Both int and char can be used in “switch” control expression.
- **C**All types i.e. int, char and float can be used in “switch” control expression.
- **D**“switch” control expression can be empty as well.

Correct Answer :B

Explanation

As per C standard, “The controlling expression of a switch statement shall have integer type.” Since char is prompted to integer in switch control expression, it’s allowed but float isn’t promoted. That’s why B is correct statement.

#619 [Explained](#) [Report](#) [Bookmark](#)

**With respect to following “for” loops in C, pick the best statement
Assume that there is a prior declaration of 'i' in all cases**

```
for (i < 10; i = 0 ; i++) // (i)  
  
for (i < 10; i++ ; i = 0) // (ii)  
  
for (i = 0; i < 10 ; i++) // (iii)  
  
for (i = 0; i++ ; i < 10) // (iv)  
  
for (i++; i = 0 ; i < 10) // (v)  
  
for (i++; i < 0 ; i = 10) // (vi)
```

- A All the above “for” loops would compile successfully.
- B All the above “for” loops would compile successfully. Except (iii), the behaviour of all the other “for” loops depend on compiler implementation.
- C Only (iii) would compile successfully
- D Only (iii) and (iv) would compile successfully

Correct Answer :A

Explanation

Basically, all of the “for” loops are valid i.e. . In the above examples, it doesn't matter what expression has been put in which part of a “for” loop. The execution order of these expressions remain same irrespective of where they have been put i.e. "1st expression" followed by "2nd expression" followed by "body of the loop" followed by "3rd expression". But the exact behavior of each of the above "for" loop depends on the body of loop as well. In fact the following is also valid and work without any issue in C. for(`printf("1st")` ; `printf("2nd")` ; `printf("3rd")`) { break; }

#620 [Explained](#) [Report](#) [Bookmark](#)

Determine Output :

```
void main()
{
    char p[] = "%dn";
    p[1] = 'c';
    printf(p, 65);
}
```

- A 65
- B c
- C A
- D ERROR

Correct Answer :C

Explanation

Due to the assignment `p[1] = 'c'` the string becomes, "%c\n". Since this string becomes the format string for printf and ASCII value of 65 is 'A', the same gets printed.

#621 [Explained](#) [Report](#) [Bookmark](#)

Determine Output :

```
#define square(x) x*x

void main()
{
    int i;

    i = 64/square(4);

    printf("%d", i);
}
```

- A4
- B64
- C16
- DNone of These

Correct Answer :B

Explanation

The macro call `square(4)` will be substituted by `4*4` so the expression becomes `i = 64/4*4`. Since / and * has equal priority the expression will be evaluated as $(64/4)*4$ i.e. $16*4 = 64$

#622 [Explained](#) [Report](#) [Bookmark](#)

What's going to happen when we compile and run the following C program?

```
#include "stdio.h"

int main()
{
    int j = 0;

    for ( ; j < 10 ; )
    {
        if (j < 10)
            printf("IB", j++);
        else
            continue;
        printf("Quiz");
    }
    return 0;
}
```

}

- ACompile Error due to use of continue in for loop.
- BNo compile error but it will run into infinite loop printing IB
- CNo compile error and it'll print IBQuiz 10 times followed by Quiz once
- DNo compile error and it'll print IBQuiz 10 times.

Correct Answer :D

Explanation

Here, initialization of j has been done outside for loop. if condition serves as control statement and prints IBQuiz 10 times due to two printf's. Please note that continue comes in picture when j becomes 10. At that time, second printf gets skipped and second expression in for is checked and it fails. Due to this, for loop ends.

#623 [Explained](#) [Report](#) [Bookmark](#)

Choose the best answer.

Prior to using a pointer variable

- A It should be declared.
- B It should be initialized.
- C It should be both declared and initialized.
- D None of these.

Correct Answer :C

Explanation

Using a pointer variable, without initializing it, will be disastrous, as it will have a garbage value.

#624 [Explained](#) [Report](#) [Bookmark](#)

Let x be an integer which can take a value of 0 or 1. The statement if($x == 0$) $x = 1$; else $x = 0$; is equivalent to which one of the following?

- A $x = 1 + x;$
- B $x = 1 - x;$
- C $x = x - 1;$
- D $x = 1 \% x;$

Correct Answer :B

Explanation

Consider (B): $x=1-x$ as we need to have x value either 0 or 1. Consider $x=0$; $x=1-0=1 \Rightarrow$ So when $x=0$ we get x value to be 1. Consider $x=1$; $x=1-1=0 \Rightarrow$ So when $x=1$ we get x value to be 0.

#625 [Explained](#) [Report](#) [Bookmark](#)

Determine Output :

```
void main()  
  
{  
  
    char far *farther, *farthest;  
  
    printf("%d..%d", sizeof(farther), sizeof(farthest));  
  
}
```

- A 4..2
- B 2..2
- C 4..4
- D 2..4

Correct Answer :A

Explanation

The second pointer is of **char** type and not a **far pointer**.

#626 [Explained](#) [Report](#) [Bookmark](#)

Assume **int** is 4 bytes, **char** is 1 byte and **float** is 4 bytes. Also, assume that pointer size is 4 bytes (i.e. typical case)

```
char *pChar;  
  
int *pInt;  
  
float *pFloat;
```

```
sizeof(pChar) ;  
  
sizeof(pInt) ;  
  
sizeof(pFloat) ;
```

What's the size returned for each of sizeof() operator?

- A 4 4 4
- B 1 4 4
- C 1 4 8
- D None of the above

Correct Answer :A

Explanation

Irrespective of the type of pointer, the size for a pointer is always same. So whether it's pointer to char or pointer to float, the size of any pointer would be same. Even size of a pointer to user defined data type (e.g. struct) is also would be same.

#627 [Explained](#) [Report](#) [Bookmark](#)

Suppose that in a C program snippet, followings statements are used.

- i) sizeof(int) ;
- ii) sizeof(int*) ;
- iii) sizeof(int**) ;

Assuming size of pointer is 4 bytes and size of int is also 4 bytes, pick the most correct answer from the given options.

- A) Only i) would compile successfully and it would return size as 4.
- B) i), ii) and iii) would compile successfully and size of each would be same i.e. 4
- C) i), ii) and iii) would compile successfully but the size of each would be different and would be decided at run time.
- D) ii) and iii) would result in compile error but i) would compile and result in size as 4.

Correct Answer :B

Explanation

Size of all pointer types is same. And whether it is a 'pointer to char' or 'pointer to int' or 'pointer to pointer to int', the size always remain same. That's why all i), ii) and iii) would compile successfully and would result in same size value of 4.

#628 [Explained](#) [Report](#) [Bookmark](#)

Determine Output :

```
void main()
{
    char *p="hi friends", *p1;
    p1=p;
```

```
while(*p != '\0') ++*p++;  
  
printf("%s", p1);  
}
```

- **A**hi friends
- **B**ij!gsjfoet
- **C**hj grjeodt
- **D**None of These

Correct Answer :B

Explanation

++*p++ will be parse in the given order : 1. *p that is value at the location currently pointed by p will be taken 2. ++*p the retrieved value will be incremented 3. when ; is encountered the location will be incremented that is p++ will be executed.

#629 [Explained](#) [Report](#) [Bookmark](#)

Array passed as an argument to a function is interpreted as

- **A** Address of the array.
- **B** Values of the first elements of the array.
- **C** Address of the first element of the array.
- **D** Number of element of the array.

Correct Answer :C

Explanation

The array passed as an argument to a function is interpreted as the address of the first element of the array.

#630 [Explained](#) [Report](#) [Bookmark](#)

Choose the best statement with respect to following three program snippets.

```
/*Program Snippet 1 with for loop*/  
  
for (i = 0; i < 10; i++)  
  
{  
  
    /*statement1*/  
  
    continue;  
  
    /*statement2*/  
  
}
```

```
/*Program Snippet 2 with while loop*/  
  
i = 0;  
  
while (i < 10)
```

```

{

    /*statement1*/

    continue;

    /*statement2*/

    i++;

}

/*Program Snippet 3 with do-while loop*/

i = 0;

do

{

    /*statement1*/

    continue;

    /*statement2*/

    i++;

}while (i < 10);

```

- **A**All the loops are equivalent i.e. any of the three can be chosen and they all will perform exactly same
- **B**continue can't be used with all the three loops in C

- **C**After hitting the continue; statement in all the loops, the next expression to be executed would be controlling expression (i.e. i < 10) in all the 3 loops
- **D**None of the above is correct

Correct Answer :D

Explanation

First and foremost, continue can be used in any of the 3 loops in C. In case of “for” loop, when continue is hit, the next expression to be executed would be i++ followed by controlling expression (i.e. i < 10). In case of “while” loop, when continue is hit, the next expression to be executed would be controlling expression (i.e. i < 10). In case of “do-while” loop, when continue is hit, the next expression to be executed would be controlling expression (i.e. i < 10). That’s why “while” and “do-while” loops would behave exactly same but not the “for” loop. Just to re-iterate, i++ would be executed in “for” loop when continue is hit. Option (D) is correct.

#631 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#include<stdio.h>

void main()
{
    int a[5] = {5, 1, 15, 20, 25};
```

```

int i, j, m;

i = ++a[1];

j = a[1]++;

m = a[i++];

printf("%d, %d, %d", i, j, m);

}

```

- A3, 2, 15
- B2, 3, 20
- C2, 1, 15
- D1, 2, 5

Correct Answer :A

Explanation

`>> int a[5] = {5, 1, 15, 20, 25};` The variable arr is declared as an integer array with a size of 5 and it is initialized to `a[0] = 5, a[1] = 1, a[2] = 15, a[3] = 20, a[4] = 25.` `>> int i, j, m;` The variable i, j, m are declared as an integer type. `>> i = ++a[1];` becomes `i = ++1;` Hence `i = 2` and `a[1] = 2` `>> j = a[1]++;` becomes `j = 2++;` Hence `j = 2` and `a[1] = 3.` `>> m = a[i++];` becomes `m = a[2];` Hence `m = 15` and i is incremented by 1(`i++` means `2++` so `i=3`) `>> printf("%d, %d, %d", i, j, m);` It prints the value of the variables i, j, m Hence the output of the program is 3, 2, 15.

#632 [Explained](#) [Report](#) [Bookmark](#)

What will happen after compiling and running following code?

```
main()  
{  
    printf("%p", main);  
}
```

- AError
- BWill make an infinite loop.
- CSome address will be printed.
- DNone of these.

Correct Answer :C

Explanation

Function names are just addresses (just like array names are addresses). `main()` is also a function. So the address of function `main` will be printed. `%p` in `printf` specifies that the argument is an address. They will be printed as hexadecimal numbers.

#633 [Explained](#) [Report](#) [Bookmark](#)

For 16-bit compiler allowable range for integer constants is
_____?

- A-3.4e38 to 3.4e38
- B-32767 to 32768
- C-32668 to 32667
- D-32768 to 32767

Correct Answer :D

Explanation

In a 16 Bit C compiler we have 2 bytes to store an integer, and 1 byte for a character. For unsigned integers the range is 0 to 65535. For signed integers the range is -32768 to 32767. For unsigned character, 0 to 255

#634 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of given program?

```
#include<stdio.h>

void main()
{
    int a=3;

    for(;a;printf("%d ", a--));
}
```

- A no output
- B 3 2 1 0
- C 3 2 1
- D infinity loop

Correct Answer :C

Explanation

Decrement operator in for loop statement are executed until the condition is true. So it is executed till "a" not equal to zero and printf statement inside for loop print a value of "a"

#635 [Explained](#) [Report](#) [Bookmark](#)

determine o/p

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

- A1
- B-2
- C2
- Derror

Correct Answer :C

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.

#636 [Explained](#) [Report](#) [Bookmark](#)

C programs are converted into machine language with the help of

- A An Editor
- B A compiler
- C An operating system
- D None of these.

Correct Answer :B

Explanation

A compiler is a system software that converts high level language into machine level language.

#637 [Explained](#) [Report](#) [Bookmark](#)

Which of the following operator takes only integer operands?

- A +
- B *
- C %
- D /

Correct Answer :C

Explanation

the modulus operator takes only integer operator

#638 Explained Report Bookmark

Determine the Final Output:

```
void main()
{
    printf("\nab");
    printf("\bsi");
    printf("\rha");
}
```

- Aabsiha
- Basiha
- chaasi
- Dhaasi

Correct Answer :D

Explanation

\n - newline - printf("\nab"); - Prints ab \b - backspace - printf("\bsi"); - firstly '\b' removes 'b' from 'ab ' and then prints 'si'. So after execution of printf

#639 [Explained](#) [Report](#) [Bookmark](#)

Determine the Final Output:

```
void main()
{
    int const *p=15;
    printf("%d", ++(*p));
}
```

- A 16
- B 15
- C Garbage Value
- D Compiler Error

Correct Answer :D

Explanation

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

#640 [Explained](#) [Report](#) [Bookmark](#)

What is right way to Initialize array?

- **A**int num[6] = { 2, 4, 12, 5, 45, 5 };
- **B**int n{} = { 2, 4, 12, 5, 45, 5 };
- **C**int n{6} = { 2, 4, 12 };
- **D**int n(6) = { 2, 4, 12, 5, 45, 5 };

Correct Answer :A

Explanation

option (B), (C) and (D) are incorrect because array declaration syntax is wrong. Only square brackets([]) must be used for declaring an array.

#641 [Explained](#) [Report](#) [Bookmark](#)

A preprocessor command

- **A** need not start on a new line
- **B** need not start on the first column
- **C** has # as the first character
- **D** need not start on the first column

Correct Answer :C

Explanation

In C programming language, preprocessor directive is a step performed before the actual source code compilation. It is not part of

the compilation. ... When we try to compile a program, preprocessor commands are executed first and then the program gets compiled. Every preprocessor command begins with # symbol.

#642 [Explained](#) [Report](#) [Bookmark](#)

What is the output of given program if user enter value 77?

```
#include <stdio.h>

void main()
{
    int i; printf("Enter a number:");
    scanf("%d", &i);
    // 77 is given as input.

    if(i%5 == 0)
    {
        printf(" Number entered is divisible by 5");
    }
}
```

- A
Enter a number:77

- **B**
Enter a number:77 Number is divisible by 5
- **C**
complier error
- **D**
Run time error

Correct Answer :A

Explanation

since this program isn't having any syntax error so program is executed. It is clearly seen that 77 is not divisible by 5. So if statement will not execute

#643 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of following program

#include

main()

{

int x,y = 10;

x = y * NULL;

printf("%d",x);

}

- A
0
- B
10
- C
error
- D
Garbage value

Correct Answer :C

Explanation

invalid operands to binary * (have 'int' and 'void *')

#644 [Explained](#) [Report](#) [Bookmark](#)

The first expression in a for loop is

- A
Step value of loop
- B
Value of the counter variable
- C
Any of above
- D
None of above

Correct Answer :B

Explanation

The initialization_expression expression executes when the loop first starts. It is typically used to initialize a loop counter variable. The

loop_condition expression is evaluated at the beginning of each iteration. The execution of the loop continues until the loop_condition evaluates to false .

#645 [Explained](#) [Report](#) [Bookmark](#)

Due to variable scope in c

- A **Variables created in a function cannot be used another function**
- B **Variables created in a function can be used in another function**
- C **Variables created in a function can only be used in the main function**
- D **None of above**

Correct Answer :A

Explanation

In C every variable defined in scope. You can define scope as the section or region of a program where a variable has its existence; moreover, that variable cannot be used or accessed beyond that region. Inside a function or a block. Out of all functions.

#646 [Explained](#) [Report](#) [Bookmark](#)

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

Correct Answer :A

Explanation

When we add a value x to a pointer p, the value of the resultant expression is $p + x * \text{sizeof}(*p)$ where $\text{sizeof}(*p)$ means size of data type pointed by p. That is why ptr2 is incremented to point to arr[3] in the above code. Same rule applies for subtraction. Note that only integral values can be added or subtracted from a pointer. We can also subtract or compare two pointers of same type.

#647 [Explained](#) [Report](#) [Bookmark](#)

Comment on the below while statement

`while (0 == 0) {}`

- A It has syntax error as there are no statements within braces {}
- B It will run forever
- C It compares 0 with 0 and since they are equal it will exit the loop immediately
- D It has syntax error as the same number is being compared with itself

Correct Answer :B

Explanation

“0==0” is always true - so this is the same as writing:

`while (true) {}` which makes the computer run in tight loop forever.

#648 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include

int main()

{

int i=1;

if(!i)

printf("SimpleWay2Code,");

else

{

i=0;

printf("C-Program");

main();

}

return 0;

}



- A prints “SimpleWay2Code, C-Program” infinitely
- B prints “C-Program” infinitely
- C prints “C-Program, SimpleWay2Code” infinitely

```

- D
Error: main() should not inside else statement

Correct Answer :B

Explanation

1. **if(!i) :** if i is false i.e. if i is 0 then if block gets executed. But in this case this if block will not get executed since i is holding 1.
2. **else block will be executed because of above statement**
3. **i=0;**

i declared as 0

`printf("C-Program");`

C-Program is printed on the screen

main() :

Main called. Since we are calling main and else block is declared as 0 it's a never ending situation i.e. there is no condition to stop the process

#649 [Explained](#) [Report](#) [Bookmark](#)

Given the below statements about C programming language;

- 1) **main() function should always be the first function present in a C program file**

- 2) all the elements of an union share their memory location
- 3) A void pointer can hold address of any type and can be typecasted to any type
- 4) A static variable hold random junk value if it is not initialised

Which of the above are correct statements?

- **A**
2,3
- **B**
1,2
- **C**
1,2,3
- **D**
1,2,3,4

Correct Answer :C

Explanation

Only Statement 4 is wrong :

Global and static variables are initialized to their default values because it is in the C or C++ standards and it is free to assign a value by zero at compile time

#650 [Explained](#) [Report](#) [Bookmark](#)

We can insert pre written code in a C program by using

- **A**
#read
- **B**
#get

- C
 #include
- D
 #put

Correct Answer :C

Explanation

In the C Programming Language, the #include directive tells the preprocessor to insert the contents of another file into the source code at the point where the #include directive is found. Include directives are typically used to include the C header files for C functions that are held outside of the current source file.

#651 [Explained](#) [Report](#) [Bookmark](#)

Output of 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

Correct Answer :B

Explanation

The function fun() expects a pointer ptr to an integer (or an address of an integer). It modifies the value at the address ptr. The dereference operator * is used to access the value at an address. In the statement ‘*ptr = 30’, value at address ptr is changed to 30. The address operator & is used to get the address of a variable of any data type. In the function call statement ‘fun(&y)’, address of y is passed so that y can be modified using its address.

#652 [Explained](#) [Report](#) [Bookmark](#)

Which operator in C can't be overloaded

- A %
- B +
- C ?:
- D -

Correct Answer :C

Explanation

The only C operators that can't be overloaded are . and ?: (and sizeof, which is technically an operator). C++ adds a few of its own operators, most of which can be overloaded except :: and .*.

#653 [Explained](#) [Report](#) [Bookmark](#)

Adding to a pointer that points to an array will

- A Cause an error
- B Increase the value of the element that the pointer is pointing to
- C Cause the pointer to point to the next element in the array
- D None of above

Correct Answer :C

Explanation

If you know that the pointer to an array element is not pointing to the last element of the array, the expression `p+1` will point to the next element of the array. This rule works regardless of the type of the array element, as long as the base type of the pointer is the same as the element type of the array: the compiler makes all the necessary adjustments when performing the addition

#654 [Explained](#) [Report](#) [Bookmark](#)

Which operator has the highest priority

- A
- ()
- B
- []
- C
- *
- D
- /

Correct Answer :A

Explanation

() operator has highest priority.

Operator associativity specifies whether, in an expression that contains multiple operators with the same precedence, an operand is grouped with the one on its left or the one on its right. Associativity is used when two operators of same precedence appear in an expression.

The associativity of an operator is a property that determines how operators of the same precedence are grouped in the absence of parentheses.

#655 Explained Report Bookmark

Memory allocation using malloc() is done in?

- A Static area
- B Stack area
- C Heap area
- D Both Stack & Heap area

Correct Answer :C

Explanation

The following declares a variable `i` on the stack:

```
int i;
```

When I ask for an address using `&i` I get the actual location on the stack.

When I allocate something dynamically using `malloc`, there are actually *TWO* pieces of data being stored. The dynamic memory is allocated on the heap, and the pointer itself is allocated on the stack. So in this code:

```
int* j = malloc(sizeof(int));
```

This is allocating space on the heap for an integer. It's also allocating space on the stack for a pointer (`j`). The variable `j`'s value is set to the address returned by `malloc`

#656 [Explained](#) [Report](#) [Bookmark](#)

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

Correct Answer :B

Explanation

Parameters are always passed by value in C. Therefore, in the above code, value of y is not modified using the function fun(). So how do we modify the value of a local variable of a function inside another function. Pointer is the solution to such problems. Using pointers, we can modify a local variable of a function inside another function. See the next question.

Note that everything is passed by value in C. We only get the effect of pass by reference using pointers.

#657 [Explained](#) [Report](#) [Bookmark](#)

What will be output of

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

```
void main()  
  
{  
  
    char test ='S';  
  
    printf("\n%c",test);  
  
}
```

- A Garbage value
- B Error
- C S
- D None of above

Correct Answer :C

Explanation

%c is used to print character

#658 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
#include <stdio.h>  
  
void main()  
  
{
```

```

int k = 5;

int *p = &k;

int **m = &p;

printf("%d%d%d\n", k, *p, **p);

}

```

- A
5 5 5
- B
5 5 junk
- C
5 junk junk
- D
Compile time error

Correct Answer :D

Explanation

`printf("%d%d%d\n", k, *p,**p);` // This line generates the error because we declare the variable `*p` and that holds the address of `k` and here we print `**p` that gives error

#659 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

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

```
void main()  
  
{  
  
    int x = 97;  
  
    int y = sizeof(x++);  
  
    printf("x is %d", x);  
  
}
```

- A
x is 97
- B
x is 98
- C
x is 99
- D
Run time error

Correct Answer :A

Explanation

`sizeof` runs at compile-time, but `x++` can only be evaluated at run-time.

To solve this, the C++ standard dictates that the operand of `sizeof` is not evaluated. The C Standard says:

so `x++` is not incremented at runtime

#660 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
#include

void main()

{

m();

void m()

{

printf("SimpleWay2Code");

}

}
```

- A SimpleWay2Code
- B Compile time error
- C Nothing
- D Varies

Correct Answer :B

Explanation

```
error: static declaration of 'm' follows non-static
declaration
```

because the function declaration is not available before the calling

#661 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
{
    int a=0;
    a = 10 + 5 * 2 * 8 / 2 + 4;
    printf("%d", a);
    return 0;
}
```

- A124
- B54
- C23
- D404

Correct Answer :B

Explanation

$$10 + 10*8/2 + 4 \quad 10 + 80/2 + 4 \quad 10 + 40 + 4 = 54$$

#662 Explained Report Bookmark

What is the output of the program.? #include<stdio.h>

```
static int k;

int main()

{

    printf("%d", k);

    return 90;

}
```

- A-1
- B0
- C90
- DCompiler error

Correct Answer :B

Explanation

Default value of a static variable is zero by default.

#663 Explained Report Bookmark

A register variable is stored in a Register. Where does a Register Present in a Computer.?

- A RAM (Random Access Memory)
- B ROM (Read Only Memory)
- C CPU (Central Processing Unit)
- D DMA (Direct Memory Access)

Correct Answer :C

Explanation

Yes. Registers are part of a CPU to store variable whose value changes frequently. Loop variables for example. register int i=1;
while(i <= 10) { printf("i=%d\n", i); }

#664 Explained Report Bookmark

Which one of the following is not a reserved keyword for C?

- A auto
- B case
- C main
- D default

Correct Answer :C

Explanation

List Of Reserved Keword for C : <https://bit.ly/3aSCuyZ>

#665 Explained Report Bookmark

```
register float a = 3.14f;
```

Choose right statement.

- A Variable a is stored in CPU registers for fast access.
- B Variable a is converted to int and then stored in a CPU register.
- C register Storage Class is ignored and treated as auto float a = 3.14f;
- D You get a compiler error as you can not store non integer value in a CPU register.

Correct Answer :C

Explanation

Yes. CPU register can not store anything more than 16 bits or 2 bytes. You will not any compiler errors. It will be treated just like an auto Storage Class variable.

#666 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program.?

```
int main()
{
    register k = 25;
    printf("%d", &k);
```

```
    return 90;  
}
```

- A Prints of address of variable k.
- B 25
- C 0
- D Compiler error

Correct Answer :D

Explanation

You can not use Ampersand & operator with a Register variable.
Register is part of CPU and accessing it is not authorized.

#667 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program statement.?

```
register int b;  
  
printf("%d", b);
```

- A null
- B 0
- C random integer number
- D random real number

Correct Answer :C

Explanation

auto and register variables hold garbage values by default.

#668 [Explained](#) [Report](#) [Bookmark](#)

What is the output the program.?

```
int main()

{
    register a=80;

    auto int b;

    b=a;

    printf("%d ", a);

    printf("%d ", b);

    return -1;
}
```

- A
Compiler error. You can not assign register value to int variable.
- B
80 80

- C
80 0
- D
Compiles, but output is none.

Correct Answer :B

Explanation

auto: This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language. Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope).

register: This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available. This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only.

Note : Both register and auto have the same functionality except the location where it is stored. Therefore the value of a is successfully copied to b. Both a and b have the same value 80 stored in it.

#669 [Explained](#) [Report](#) [Bookmark](#)

The statement below isa

```
extern int p;
```

- ADeclaration

- **B**Definition
- **C**Initialization
- **D**None of the above

Correct Answer :A

Explanation

Usually, most of the C statements with extern keyword are Declarations.

#670 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program.?

```
int main()

{
    auto int a=10;

    {
        auto int a = 15;

        printf("%d ", a);

    }

    printf("%d ", a);

    return 1;
}
```

}

- A10 10
- B10 15
- C15 10
- DCompiler error

Correct Answer :C

Explanation

Automatic or Register variables have block scope and life. Most recent definition takes precedence. {...} is a block.

#671 Explained Report Bookmark

What is the output of the C statement.?

```
int main()
{
    int a=0;
    a = 4 + 4/2*5 + 20;
    printf("%d", a);
    return 0;
}
```

- A 40
- B 4
- C 34
- D 54

Correct Answer :C

Explanation

/ and * has equal priority. But associativity is from L to R.
 $4 + 2 * 5 + 20 = 34$

#672 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program.?

```
void myshow();
```

```
int main()
```

```
{
```

```
    myshow();
```

```
    myshow();
```

```
    myshow();
```

```
}
```

```
void myshow()

{

    static int k = 20;

    printf("%d ", k);

    k++;

}
```

- A 20 20 20
- B 20 21 21
- C 20 21 22
- D Compiler error.

Correct Answer :C

Explanation

Variable of type static holds its value until the end of program execution. Static variables do not initialize again and again with function calls.

#673 [Explained](#) [Report](#) [Bookmark](#)

What is a C Storage Class.?

- A
C Storage decides where to or which memory store the variable.

- **B**
C Storage Class decides what is the default value of a variable.
- **C**
C Storage Class decides what is the Scope and Life of a variable.
- **D**
All the above.

Correct Answer :D

Explanation

In C language, each variable has a storage class which decides the following things:

- scope i.e where the value of the variable would be available inside a program.
- default initial value i.e if we do not explicitly initialize that variable, what will be its default initial value.
- lifetime of that variable i.e for how long will that variable exist.

#674 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
int main()
```

```
{
```

```
    int a=9;
```

```
    if(a==9);
```

```
{  
  
    printf("Ostrich\n");  
  
}  
  
elseif(a==8)  
  
{  
  
    printf("Eggs\n");  
  
}  
  
printf("White");  
  
return 0;  
}
```

- A White
- B Ostrich White
- C No Ouput
- D Compiler error

Correct Answer :D

Explanation

Notice IFELSE statement. There should be one SPACE between IF and ELSE.

#675 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
int main()
{
    int a=9;
    if(a==8)
    {
        printf("Kangaroo\n");
    }
    printf("Eggs\n");
    return 0;
}
```

- A No output
- B Eggs
- C Kangaroo Eggs
- D Compiler error

Correct Answer :C

Explanation

a=8 is an assignment not comparison. IF(Non Zero) is always TRUE.

#676 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
{
    if( 4 > 5 )
        printf("Hurray..\n");
    printf("Yes");
    return 0;
}
```

- A Yes
- B Hurray.. Yes
- C Hurray..Yes
- D No Output

Correct Answer :A

Explanation

To include more than one statement inside If block, use { } braces. Otherwise, only first statement after if block is included. IF condition fails with false. So second if which is outside of If is executed.

#677 [Explained](#) [Report](#) [Bookmark](#)

What is the Priority among (*, /, %), (+, -) and (=) C Operators.?

- A(*, /, %) > (+, -) < (=)
- B(*, /, %) < (+, -) < (=)
- C(*, /, %) > (+, -) > (=)
- D(*, /, %) < (+, -) (+, -) == (=)

Correct Answer :C

Explanation

Assignment operator in C has the least priority.

#678 [Explained](#) [Report](#) [Bookmark](#)

A C variable name can start with a _

- A
Number
- B
Plus Sign (+)
- C
Underscore

- D
Asterisk (*)

Correct Answer :C

Explanation

The first character must be a letter or an underscore (_). You can't use a number as the first character. The rest of the variable name can include any letter, any number, or the underscore. You can't use any other characters, including spaces, symbols, and punctuation marks

#679 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with functions.?

```
void main()
{
    int a;
    printf("TIGER COUNT=");
    a=show();
    printf("%d", a);
}

int show()
```

```
{  
  
    return 15;  
  
    return 35;  
  
}  
  


- A TIGER COUNT=15
- B TIGER COUNT=35
- C TIGER COUNT=0
- D Compiler error

```

Correct Answer :A

Explanation

More than one return statement will not cause Compiler Error. But only FIRST return STATEMENT is executed. Anything after return 15; is not reachable.

#680 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
void show();  
  
void main()  
{
```

```
    printf("PISTA ") ;

    show() ;

}
```

```
void show()

{

    printf("CACHEW ") ;

    return 10 ;

}
```

- A PISTA CACHEW
- B CASHEW PISTA
- C PISTA CASHEW with compiler warning
- D Compiler error

Correct Answer :C

Explanation

void show() function should not return anything. So return 10; is not recommended.

#681 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()

{
    int a= 3 + 5/2;

    printf("%d", a);

    return 0;
}
```

- A3
- B2
- C5
- DCan not assign an expression to variable at the time of declaration.

Correct Answer :C

Explanation

Assignment Operator = in C language has the least priority. So the right hand side expression is evaluated first and then assigned to the left side variable. $a = 3 + 5/2; a = 3 + 2; a = 5;$

#682 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with functions.?

```
int main()
```

```
{  
  
    show();  
  
    printf("BANK ");  
  
    return 0;  
  
}
```

```
void show()  
  
{  
  
    printf("CURRENCY ");  
  
}
```

- A CURRENCY BANK
- B BANK CURRENCY
- C BANK
- D Compiler error

Correct Answer :D

Explanation

Yes. Compiler error. Before calling the show(); function, its Function Prototype should be declared before outside of main() and before main(). void show(); int main() { show(); printf("BANK "); return 0; }

#683 [Explained](#) [Report](#) [Bookmark](#)

How many values can a C Function return at a time.?

- A Only One Value
- B Maximum of two values
- C Maximum of three values
- D Maximum of 8 values

Correct Answer :A

Explanation

Using a return val; statement, you can return only one value.

#684 [Explained](#) [Report](#) [Bookmark](#)

Name the loop that executes at least once.

- A For
- B If
- C do-while
- D while

Correct Answer :C

Explanation

In most computer programming languages, a do while loop is a control flow statement that executes a block of code at least once,

and then either repeatedly executes the block, or stops executing it, depending on a given boolean condition at the end of the block.

#685 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
int myshow(int);
```

```
void main()
```

```
{
```

```
    myshow(5);
```

```
    myshow(10);
```

```
}
```

```
int myshow(int b)
```

```
{
```

```
    printf("Received %d, ", b);
```

```
}
```

- A Received 5, Received 10,
- B Received 10, Received 5,
- C Received 0, Received 0,

- **DCompiler error**

Correct Answer :A

Explanation

Notice the function prototype declaration int myshow(int). If you declare wrong either Compiler warning or error is thrown. myshow(5) passes number 5. 5 is received as variable int b.

#686 [Explained](#) [Report](#) [Bookmark](#)

Prototype of a function means _____

- **A Name of Function**
- **B Output of Function**
- **C Declaration of Function**
- **D Input of a Function**

Correct Answer :C

Explanation

In computer programming, a function prototype or function interface is a declaration of a function that specifies the function's name and type signature, but omits the function body

#687 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```

int main()

{

    int a=0;

    a = 10 + 2 * 12 / 3 * 2 + 5;

    printf("%d", a);

    return 0;

}

```

- A19
- B31
- C11
- D25

Correct Answer :B

Explanation

$10 + 2 * 12 / 3 * 2 + 5; 10 + 24/3*2 + 5; 10 + 8*2 + 5; 10 + 16 + 5 = 31;$

#688 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
int main()
```

```
{  
  
    if("abc")  
  
    {  
  
        printf("India\n");  
  
    }  
  
    if('c')  
  
    {  
  
        printf("Honey\n");  
  
    }  
  
    printf("ZING");  
  
  
    return 0;  
}
```

- A ZING
- B Honey ZING
- C India ZING
- D India Honey ZING

Correct Answer :D

Explanation

"abc" is string and it returns an Integer Address Number. 'c' returns an ASCII number which is also a number. Any Non-Zero number gives TRUE condition.

#689 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with functions.?

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

```
int show();
```

```
void main()
```

```
{
```

```
    int a;
```

```
    a=show();
```

```
    printf("%d", a);
```

```
}
```

```
int show() {
```

```
    return 15.5;
```

```
    return 35;
```

}

- A
15.5
- B
0
- C
15
- D
Compiler error

Correct Answer :C

Explanation

It is perfectly Okay to return a float number 15.5 as an Integer inside int show() function. 15.5 is demoted to integer as 15 and returned.

#690 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with Functions.?

```
int main()
{
    void show()
    {
        printf("HIDE");
    }
}
```

```
    }

    show();

    return 0;
}



- A No output
- B HIDE
- C Compiler error
- D None of the above

```

Correct Answer :B

Explanation

Notice that `show()` function is defined inside `main()` function. It will not produce a compile error. But, it is not recommended to define a **FUNCTION INSIDE A FUNCTION. DO NOT DO.**

#691 [Explained](#) [Report](#) [Bookmark](#)

Determine Output:

```
main()
```

```
{
```

```

char *str1 = "abcd";

char str2[] = "abcd";

printf("%d %d %d", sizeof(str1), sizeof(str2),
sizeof("abcd"));

}

• A2 5 5
• B2 4 4
• C8 5 5
• D2 4 5

```

Correct Answer :C

Explanation

In first sizeof, str1 is a character pointer so it gives you the size of the pointer variable. In second sizeof the name str2 indicates the name of the array whose size is 5 (including the 'null' termination character). The third sizeof is similar to the second one.

#692 [Explained](#) [Report](#) [Bookmark](#)

Any C program

- A Must contain at least one function.
- B Need not contain any function.
- C Needs input data.
- D None of the above

Correct Answer :A

Explanation

At least one function which must be included in any C program is **main()**.

#693 [Explained](#) [Report](#) [Bookmark](#)

Determine output:

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

- A 6
- B 5
- C Garbage Value
- D Compiler Error

Correct Answer :D

Explanation

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

#694 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```

#include<stdio.h>

void main()

{
    int a[5] = {5, 1, 15, 20, 25};

    int i, j, m;

    i = ++a[1];

    j = a[1]++;

    m = a[i++];

    printf("%d, %d, %d", i, j, m);

}

```

- A3, 2, 15
- B2, 3, 20
- C2, 1, 15
- D1, 2, 5

Correct Answer :A

Explanation

>> int a[5] = {5, 1, 15, 20, 25}; The variable arr is declared as an integer array with a size of 5 and it is initialized to a[0] = 5, a[1] = 1, a[2] = 15,

`a[3] = 20, a[4] = 25.` `>> int i, j, m;` The variable i, j, m are declared as an integer type. `>> i = ++a[1];` becomes `i = ++1;` Hence `i = 2` and `a[1] = 2` `>> j = a[1]++;` becomes `j = 2++;` Hence `j = 2` and `a[1] = 3.` `>> m = a[i++];` becomes `m = a[2];` Hence `m = 15` and i is incremented by 1(`i++` means `2++` so `i=3`) `>> printf("%d, %d, %d", i, j, m);` It prints the value of the variables i, j, m Hence the output of the program is 3, 2, 15.

#695 [Explained](#) [Report](#) [Bookmark](#)

What is the output of given program if user enter "xyz" ?

```
#include

void main()
{
    float age, AgeInSeconds;

    printf("Enter your age:");

    scanf("%f", &age);

    AgeInSeconds = 365 * 24 * 60 * 60 * age;

    printf("You have lived for %f seconds", AgeInSeconds);

}
```

- A Enter your age: xyz You have lived for 0 seconds

- **B**Enter your age: xyz You have lived for 0.00000 seconds
- **C**Enter your age: xyz "after that program will stop"
- **D**Run time error

Correct Answer :B

Explanation

When we give `scanf()` a "%f" format string, that means "We want you to try and get us a floating point number. When we provide input like 'xyz', it's not going to match anything, because 'xyz' is not a valid floating-point number.

#696 [Explained](#) [Report](#) [Bookmark](#)

Determinine o/p

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

- **A**5 5 5 5 5

- **B** 5 4 3 2 1
- **C** Infinite Loop
- **D** None of These

Correct Answer :B

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.

#697 [Explained](#) [Report](#) [Bookmark](#)

Determine Output:

```
void main()
{
    char far *farther, *farthest;
    printf("%d..%d", sizeof(farther), sizeof(farthest));
}
```

- **A** 4..2
- **B** 2..2
- **C** 4..4

- D2..4

Correct Answer :A

Explanation

The second pointer is of char type and not a far pointer.

#698 [Explained](#) [Report](#) [Bookmark](#)

Determine output:

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

- Ammm nnn aaa
- Bmmmm nnnn aaaa
- CCompiler Error
- DNone of These

Correct Answer :D

Explanation

Correct Output : mmmm aaaa nnnn 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].

#699 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the given program?

```
#include<stdio.h>

void main()
{
    int  i=10;
    printf("i=%d", i);

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

```
    printf("i=%d", i);

}

printf("i=%d", i);
```

- A 10 10 11 11
- B 10 20 21 21
- C 10 20 21 10
- D 10 20 21 20

Correct Answer :C

Explanation

The scope of second declaration of i is limited to the block in which it is defined. Outside of the block variable is not recognized.

#700 [Explained](#) [Report](#) [Bookmark](#)

```
char *ptr;

char myString[] = "abcdefg";

ptr = myString;

ptr += 5;
```

what string does ptr point to in the sample code above?

- Afg
- Befg
- Cdefg
- Dcdefg

Correct Answer :A

Explanation

NnN

#701 [Explained](#) [Report](#) [Bookmark](#)

Which of the following function is used to find the first occurrence of a given string in another string?

- Astrchr()
- Bstrrchr()
- Cstrstr()
- Dstrnset()

Correct Answer :C

Explanation

The function strstr() Finds the first occurrence of a substring in another string Declaration: char *strstr(const char *s1, const char *s2); Return Value: On success, strstr returns a pointer to the element

in s1 where s2 begins (points to s2 in s1). On error (if s2 does not occur in s1), strstr returns null.

#702 [Explained](#) [Report](#) [Bookmark](#)

What will happen after compiling and running following code?

```
main()
{
    printf("%p", main);
}
```

- A Error
- B Will make an infinite loop.
- C Some address will be printed.
- D None of these.

Correct Answer :C

Explanation

Function names are just addresses (just like array names are addresses). main() is also a function. So the address of function main will be printed. %p in printf specifies that the argument is an address. They will be printed as hexadecimal numbers.

#703 [Explained](#) [Report](#) [Bookmark](#)

What will be the value of i and j after execution of following program?

```
#include<stdio.h>

void main()
{
    int i, j;

    for(i=0, j=0; i<10, j<20; i++, j++) {
        printf("i=%d %t j=%d", i, j);
    }
}
```

- A 10 10
- B 10 20
- C 20 20
- D Run time error

Correct Answer :C

Explanation

comma operator is executed from left to right so until $j < 20$ for loop statement is true, so both i and j are incremented. So, $i = 20$ and $j = 20$.

#704 [Explained](#) [Report](#) [Bookmark](#)

Determinine o/p

```
void main()
{
    int i=i++, j=j++, k=k++;
    printf("%d %d %d", i, j, k);
}
```

- A 1 1 1
- B 0 0 0
- C garbage values
- D Error

Correct Answer :C

Explanation

An identifier is available to use in program code from the point of its declaration. So expressions such as `i = i++` are valid statements. The `i`, `j` and `k` are automatic variables and so they contain some garbage value.

#705 [Explained](#) [Report](#) [Bookmark](#)

Determine output:

```
void main()
{
```

```

float me = 1.1;

double you = 1.1;

if(me==you)

    printf("I hate Exam");

else

    printf("I love Exam");

}

```

- **A**I hate Exam
- **B**I love Exam
- **C**Error
- **D**None of These

Correct Answer :B

Explanation

For floating point numbers (float, double, long double) the values cannot be predicted exactly. Depending on the number of bytes, the precision with 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 (== , >, <, <=, >=, !=)

#706 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```

#include <stdio.h>

int main(void)

{

    char p;

    char buf[10] = {1, 2, 3, 4, 5, 6, 9, 8};

    p = (buf + 1)[5];

    printf("%d", p);

    return 0;
}

```

- A5
- B6
- C9
- Derror

Correct Answer :C

Explanation

x[i] is equivalent to ***(x + i)**, so **(buf + 1)[5]** is ***(buf + 1 + 5)**, i.e. **buf[6]**.

#707 [Explained](#) [Report](#) [Bookmark](#)

Determine Output:

```
#include<stdio.h>

#define a 10

void main()

{

#define a 50

printf("%d", a);

}
```

- A 50
- B 10
- C Compiler Error
- D None of These

Correct Answer :A

Explanation

The preprocessor directives can be redefined anywhere in the program. So the most recently assigned value will be taken.

#708 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```

#include

void main()

{

    int y=10;

    if(y++>9 && y++!=10 && y++>11)

        printf("%d", y);

    else

        printf("%d", y);

}

```

- A11
- B12
- C13
- D14

Correct Answer :C

Explanation

All the three condition in if is true. if($y++ > 9 \ \&\& \ y++ \neq 10 \ \&\& \ y++ > 11$)
such as 1st condition : $10++ > 9$ is true and value of y is increase by 1
i.e y =11 2nd condition : $11++ \neq 10$ is also ture and value of y is
increase by 1 i.e y =12 3rd condition : $12++ > 11$ is also ture and value

of y is increase by 1 i.e y =13 Therefore if is excuted and print the value of y = 13

#709 [Explained](#) [Report](#) [Bookmark](#)

Choose the best answer. Prior to using a pointer variable

- A It should be declared.
- B It should be initialized.
- C It should be both declared and initialized.
- D None of these.

Correct Answer :C

Explanation

Using a pointer variable, without initializing it, will be disastrous, as it will have a garbage value.

#710 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

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

- A1
- B-2
- C2
- Derror

Correct Answer :C

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.

#711 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#include

#define int char

void main()

{
    int i = 65;

    printf("sizeof(i)=%d", sizeof(i));
}
```

- A `sizeof(i)=2`
- B `sizeof(i)=1`
- C Compiler Error
- D None of These

Correct Answer :B

Explanation

Since the `#define` replaces the string `int` by the macro `char`. So, here `i` is a variable of type `char` and not `int`.

#712 Explained Report Bookmark

Determine Output:

```
void main()
{
    char *p="hi friends", *p1;
    p1=p;
    while(*p!='\0') ++*p++;
    printf("%s", p1);
}
```

- A hi friends
- B ij!gsjfoet

- C hj grjeodt
- D None of These

Correct Answer :B

Explanation

`++*p++` will be parse in the given order :

1. *p that is value at the location currently pointed by p will be taken
2. ++*p the retrieved value will be incremented
3. when ; is encountered the location will be incremented that is `p++` will be executed

#713 [Explained](#) [Report](#) [Bookmark](#)

Which operator from the following has the lowest priority?

- A Assignment operator
- B Division operator
- C Comma operator
- D Conditional operator

Correct Answer :C

Explanation

Ref C Operator Precedence Table

#714 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#define INC(X) X++  
  
void main()  
  
{  
  
    int x=4;  
  
    printf("%d", INC(x++));  
  
}
```

- A4
- B5
- C6
- Derror

Correct Answer :D

Explanation

L value is required for this expression (x++) which is illegal.

#715 [Explained](#) [Report](#) [Bookmark](#)

Let x be an array. Which of the following operations are illegal?

- I. ++x

II. $x+1$

III. $x++$

IV. x^2

- A I and II
- B I, II and III
- C II and III
- D I, III and IV

Correct Answer :D

Explanation

`int x[10]; * x will store the base address of array. *`

Statement I, III and IV is invalid.

**Statement I and III : $++x$ and $x++$ are throwing en error while compile
(lvalue required as increment operand)**

**Since, x is storing in the address of the array which is static value
which cannot be change by the operand.**

**Statement IV : x^2 is also throw an error while compile (invalid
operands to binary * (have 'int *' and 'int'))**

Statement II : x+1 is throw a warning: assignment makes integer from pointer without a cast [enabled by default]

#716 [Explained](#) [Report](#) [Bookmark](#)

What is right way to Initialize array?

- Aint num[6] = { 2, 4, 12, 5, 45, 5 };
- Bint n{} = { 2, 4, 12, 5, 45, 5 };
- Cint n{6} = { 2, 4, 12 };
- Dint n(6) = { 2, 4, 12, 5, 45, 5 };

Correct Answer :A

Explanation

option (B), (C) and (D) are incorrect because array declaration syntax is wrong. Only square brackets([]) must be used for declaring an array.

#717 [Explained](#) [Report](#) [Bookmark](#)

Assuming a short is two bytes long, what will be printed by the given code?

```
short testarray[4][3] = { {1}, {2,3}, {4,5,6}};  
printf("%d", sizeof(testarray));
```

- A6
- B7
- C12
- D24

Correct Answer :D

Explanation

No spaces are allowed in the variable names.

The following table provides the details of standard integer types with their storage sizes and value ranges –

TYPE	STORAGE SIZE	VALUE RANGE
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255

signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

#718 [Explained](#) [Report](#) [Bookmark](#)

Determine Output:

```
void main()
{
    char not;
    not = !2;
    printf("%d", not);
}
```

- A 2
- B Garbage Value
- C 0
- D None of These

Correct Answer :C

Explanation

`!` is a logical operator. In C the value 0 is considered to be the boolean value FALSE, and any non-zero value is considered to be the boolean value TRUE. Here 2 is a non-zero value so TRUE. `!TRUE` is FALSE (0) so it prints 0.

#719 [Explained](#) [Report](#) [Bookmark](#)

Determine Output:

```
void main()
{
    int i=5;

    printf("%d%d%d%d", i++, i--, ++i, --i, i);
}
```

- A45545
- B54544
- C55445
- D54554

Correct Answer :A

Explanation

The arguments in a function call are pushed into the stack from left to right. The evaluation is by popping out from the stack. and the evaluation is from right to left, hence the result.

#720 [Explained](#) [Report](#) [Bookmark](#)

Which of following is not a valid name for a C variable?

- AInfoBeam_ITS
- BInfoBeam
- CInfo Beam

- DBoth A and B

Correct Answer :C

Explanation

No spaces are allowed in the variable names.

#721 [Explained](#) [Report](#) [Bookmark](#)

Range of signed char and unsigned char are.?

- A-128 to +127, 0 to 255
- B0 to 255, -128 to +127
- C-128 to -1, 0 to +127
- D0 to +127, -128 to -1

Correct Answer :A

Explanation

Advantage of an unsigned representation is only to increase the upper limit i.e positive limit. Size of a char remains same i.e 1 Byte.

#722 [Explained](#) [Report](#) [Bookmark](#)

Types of Integers are.?

- Ashort
- Bint
- Clong
- DAll the above

Correct Answer :D

Explanation

Size of int < long.

#723 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
int main()
{
    int a=0, b;
    a = (5>2) ? b=6: b=8;
    printf("%d %d",a, b);

    return 0;
}
```

- A 6 6
- B 0 6
- C 0 8
- D compiler error

Correct Answer :D

Explanation

Compiler error. a = (5>2) ? b=6: b=8; should be written as a = (5>2) ? b=6: (b=8); main.c: In function ‘main’: main.c:14:23: error: l value required as left operand of assignment a = (5>2) ? b=6: b=8;

#724 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program.?

```
int main()
{
    int a=0;
    a = printf("4");
    printf("%d",a);
    return 0;
}
```

- A04
- Bcompiler error
- C40
- D41

Correct Answer :D

Explanation

a = printf("4"); First printf prints 4. printf() returns 1. Now the variable a=1; So 1 is printed next.

#725 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
{
    if( 4 > 5 )
    {
        printf("Hurray..\n");
    }
    printf("Yes");
}

return 0;
}
```

- A Yes
- B Hurray.. Yes
- C Hurray..Yes
- D Compiler error

Correct Answer :A

Explanation

if condition fails. So control will not enter Hurray printf statement.

#726 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
{
    if(10.0)
    {
        printf("Texas\n");
    }
    printf("ZING");
    return 0;
}
```

- A ZING
- B Texas ZING
- C Compiler error.

- **D**None of the above.

Correct Answer :B

Explanation

You can use either Integer or Real numbers. 0 or 0.0 evaluates to false condition.

#727 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
{
    if( 10 < 9 )
        printf("Hurray..\n");
    else if(4 > 2)
        printf("England");
    return 0;
}
```

- **A**England
- **B**Hurray..

- **CCompiler error for missing else**
- **DNone of the above**

Correct Answer :A

Explanation

You can omit ELSE comfortably. Compiler will not complain above ELSE after IF or ELSE IF.

#728Explained Report Bookmark

What is the output of the C Program.?

```
int main()
{
    int a=0;
    a = 5>2 ? printf("4") : 3;
    printf("%d",a);
    return 0;
}
```

- **Acompiler error**
- **B14**
- **C41**

- D0

Correct Answer :C

Explanation

5>2 is true. So expression1 i.e printf("4) is executed printing 4. Function printf() returns 1. So a value is 1

#729 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
{
    int a=0;
    a = (5>2) ? : 8;
    printf("%d",a);
    return 0;
}
```

- A0
- B1
- C8
- Dcompiler error

Correct Answer :B

Explanation

expression1 = empty expression2 = 8 If no expression is specified, it will be treated as 1.

#730 [Explained](#) [Report](#) [Bookmark](#)

Identify wrong C Keywords below.

- Aunion, const, var, float
- Bshort, unsigned, continue, for
- Csigned, void, default, goto
- Dsizeof, volatile, do, if

Correct Answer :A

Explanation

'var' is not a valid keyword.

#731 [Explained](#) [Report](#) [Bookmark](#)

Choose a correct statement regarding C Comparison Operators.

- A $(x == y)$ Is x really equal to y. $(x != y)$ Is x not equal to y.
- B $(x < y)$ Is x less than y $(x > y)$ Is x greater than y
- C $(x <= y)$ Is x less than or equal to y. $(x >= y)$ Is x greater than or equal to y

- D
All the above

Correct Answer :D

Explanation

OPERATOR	DESCRIPTION	EXAMPLE
<code>==</code>	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	$(A == B)$ is not true.
<code>!=</code>	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	$(A != B)$ is true.
<code>></code>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	$(A > B)$ is not true.
<code><</code>	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	$(A < B)$ is true.
<code>>=</code>	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	$(A >= B)$ is not true.

<=	<p>Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.</p>	<p>(A <= B) is true.</p>
----	---	------------------------------------

#732 **Explained** [Report](#) [Bookmark](#)

What is the output of the C statement.?

```
int main()
```

```
{
```

```
int a=0;
```

```
a = 5<2 ? 4 : 3;
```

```
printf("%d",a);
```

```
return 0;
```

```
}
```

- A4
- B3
- C5
- D2

Correct Answer :B

Explanation

5<2 is false. So 3 will be picked and assigned to the variable a.

#733 [Explained](#) [Report](#) [Bookmark](#)

signed and unsigned representation is available for.?

- A short, int, long, char
- B float, double, long double
- C A & B
- D None of the above

Correct Answer :C

Explanation

Real numbers like float, double and long double do not support unsigned representation.

#734 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the program.? #include<stdio.h>

```
int main()
{
    printf("Hello Boss.");
}
```

- A Hello Boss.
- B hello boss
- C No output
- D Compiler error

Correct Answer :D

Explanation

Notice the return type int before main() method. So main should maintain a return 0; or return somenumber; statement.

#735 [Explained](#) [Report](#) [Bookmark](#)

Choose a C Conditional Operator from the list.

- A ?:
- B :?
- C :<
- D <:

Correct Answer :A

Explanation

? : = Question Mark Colon is also called C Ternary Operator.

#736 [Explained](#) [Report](#) [Bookmark](#)

Property which allows to produce different executable for different platforms in C is called?

- A File inclusion
- B Selective inclusion
- C Conditional compilation
- D Recursive macros

Correct Answer :C

Explanation

Conditional compilation is the preprocessor facility to produce different executable.

#737 [Explained](#) [Report](#) [Bookmark](#)

What does fp point to in the program ?

```
#include<stdio.h>

int main()

{
    FILE *fp;

    fp=fopen("trial", "r");

    return 0;
}
```

- A The first character in the file
- B A structure which contains a char pointer which points to the first character of a file.
- C The name of the file.
- D The last character in the file.

Correct Answer :B

Explanation

The fp is a structure which contains a char pointer which points to the first character of a file.

#738 [Explained](#) [Report](#) [Bookmark](#)

C preprocessors can have compiler specific features.

- Atrue
- Bfalse
- CDepends on the platform
- DDepends on the standard

Correct Answer :A

Explanation

#pragma is compiler specific feature.

#739 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code? What is the output of this C code?

```
int main()

{
    int i = 10;

    void *p = &i;

    printf("%d\n", (int)*p);

    return 0;
```

}

- A Compile time error
- B Segmentation fault/runtime crash
- C 10
- D Undefined behaviour

Correct Answer :A

Explanation

`printf("%d\n", (int)*p);`, it will give compile time error, you can not dereference void pointer.

#740 [Explained](#) [Report](#) [Bookmark](#)

Which of the following does not initialize ptr to null (assuming variable declaration of a as int a=0)?

- A `int *ptr = &a;`
- B `int *ptr = &a - &a;`
- C `int *ptr = a - a;`
- D All of the mentioned

Correct Answer :A

Explanation

Here *ptr is initialized with address of variable a and not with NULL.

#741 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
int main()
{
    int i = -5;
    int k = i %4;
    printf("%d\n", k);
}
```

- A
Compile time error
- B
-1
- C
1

- D
None

Correct Answer :B

Explanation

```
#include <stdio.h>

int main()

{

    int i = -5;

    int k = i % 4; // here k = -5 % 4 will give a -1

    printf("%d\n", k);

}
```

#742 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
int main()
```

```
{
```

```
    int i = 5;
```

```
    int l = i / -4;
```

```
    int k = i % -4;
```

```
    printf("%d %d\n", l, k);
```

```
    return 0;
```

```
}
```

- A
Compile time error
- B
-1 1
- C
1 -1
- D
Run time error

Correct Answer :B

Explanation

int l = i/-4 give 5/-4 which is -1

int l = i/-4 give 5%-4 which is 1 (because always numerator sign is considered during modulus operation)

#743 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
void main()
```

```
{
```

```
    int x = 4.3 % 2;
```

```
    printf("Value of x is %d", x);
```

```
}
```

- A
Value of x is 1.3
- B
Value of x is 2
- C
Value of x is 0.3
- D
Compile time error

Correct Answer :D

Explanation

The modulus operator only works on integers. For floating point values use `fmod` or `fmodf`.

o you will get

```
error: invalid operands to binary % (have 'double' and 'int')
```

#744 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
int main()
{
    int x = 2, y = 0;
    int z = (y++) ? y == 1 && x : 0;
    printf("%d\n", z);
    return 0;
}
```

- **A**
0
- **B**
1
- **C**
Undefined behavior
- **D**
Compile time error

Correct Answer :A

Explanation

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

```
int main()

{

    int x = 2, y = 0;

    int z = (y++) ? y == 1 && x : 0;

    //here y = 0 so in ternary operator it false

    //now so z = 0 value assigned then value of y++
    incremented

    printf("%d\n", z);

    return 0;

}
```

#745 [Explained](#) [Report](#) [Bookmark](#)

What is the output of this C code?

```
int main()
```

```
{
```

```
int y = 1, x = 0;  
  
int l = (y++, x++) ? y : x;  
  
printf("%d\n", l);  
  
}
```

- **A**
1
- **B**
2
- **C**
Compile time error
- **D**
Undefined behaviour

Correct Answer :A

Explanation

first it will checks whether (y++,x++) is true or not

$(y++,x++) = (1,0)$

=1 which is true

Hence it going to execute y's value

i.e y=1

remember: if it returns false or 0 then it returns x's value

#746 [Explained](#) [Report](#) [Bookmark](#)

o print out a and b given below, which of the following printf() statement will you use?

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

```
float a=3.14;
```

```
double b=3.14;
```

- Aprintf("%f %lf", a, b);
- Bprintf("%Lf %f", a, b);
- Cprintf("%Lf %Lf", a, b);
- Dprintf("%f %Lf", a, b);

Correct Answer :A

Explanation

To print a float value, %f is used as format specifier. To print a double value, %lf is used as format specifier. Therefore, the answer is printf("%f %lf", a, b);

#747 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with Structure pointer in TurboC.?

```
int main()

{

    struct books{

        int pages;

        char str[4];

    }*ptr;

    printf("%d",sizeof(ptr));

    return 0;

}
```

- A2
- B6
- C7
- D8

Correct Answer :B

Explanation

Memory reserved will be the size of sum of individual elements.

#748 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with structures.?

```
void show(int,int);
```

```
int main()
```

```
{
```

```
    struct paint{
```

```
        int type;
```

```
        int color;
```

```
    }p;
```

```
    p.type=1;
```

```
    p.color=5;
```

```
    show(p.type,p.color);
```

```
    return 0;
```

```
}
```

```
void show(int a,int b)
```

```
{
```

```
    printf("%d %d",a,b);
```

```
}
```

- A1 1
- B1 5
- C5 1
- DCompiler error

Correct Answer :B

Explanation

NaN

#749 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with switch statement or block.?

```
int main()

{
    char code='K';

    switch(code)

    {
        case 'A': printf("ANT ");break;

        case 'K': printf("KING "); break;

        default: printf("NOKING");
    }
}
```

```
    }  
  
    printf("PALACE");  
  
}
```

- A KING PALACE
- B KING NOTHING PALACE
- C ANT KING PALACE
- D Compiler error for using Non Integers as CASE constants.

Correct Answer :A

Explanation

Any character is replaced by ASCII code or number by the compiler.
So it is allowed to use CHARACTER constants for CASE constants.

#750 [Explained](#) [Report](#) [Bookmark](#)

Choose the right statement.

```
int main()  
  
{  
  
    float c = 3.5 + 4.5;  
  
    printf("%f", c);
```

```
    return 0;  
}
```

- A
8.0
- B
8.000000
- C
8
- D
7

Correct Answer :B

Explanation

Float can print precision up to 6 digits. So 6 zeros will be shown if there are no digits after decimal point.

#751 [Explained](#) [Report](#) [Bookmark](#)

Choose a correct statement about C structure elements.?

- A Structure elements are stored on random free memory locations
- B Structure elements are stored in register memory locations
- C Structure elements are stored in contiguous memory locations
- D None of the above.

Correct Answer :C

Explanation

NaN

#752 Explained Report Bookmark

What is the output of C Program with switch statement or block.?

```
int main()

{
    char code='K';

    switch(code)

    {
        case "A": printf("ANT ") ;break;

        case "K": printf("KING ") ; break;

        default: printf("NOKING");

    }

    printf("PALACE");
}
```

- AANT KING PALACE
- BKING PALACE
- CPALACE
- DCompiler error

Correct Answer :D

Explanation

You can not use STRING constants with DOUBLE QUOTES as CASE constants. Only expressions leading or constants leading to Integers are allowed.

#753 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```
int main()

{
    int a;

    switch(a)

    {
        printf("APACHE ");
    }
}
```

```
    printf("HEROHONDA") ;  
}
```

- A APACHE HEROHONDA
- B HEROHONDA
- C No Output
- D Compiler error

Correct Answer :B

Explanation

Notice the missing CASE or DEFAULT statements. Still compiler accepts. But without CASE statement nothing will be printed inside of SWITCH.

```
switch(a) { printf("APACHE "); }
```

#754 [Explained](#) [Report](#) [Bookmark](#)

Choose a right statement.

```
float var = 3.5 + 4.5;
```

- A var = 8.0
- B var = 8
- C var = 7
- D var = 0.0

Correct Answer :A

Explanation

A float variable can hold a real number.

#755 [Explained](#) [Report](#) [Bookmark](#)

Choose a right statement.

```
int a = 3.5 + 4.5;
```

- Aa = 0
- Ba = 7
- Ca = 8
- Da = 8.0

Correct Answer :C

Explanation

3.5 + 4.5 = 8.0 is a real number. So it is converted to downgraded to int value. So a = 8.

#756 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```
int main()
{
    char code='A';
    switch(code)
```

```
{  
  
    case 64+1: printf("ANT ");break;  
  
    case 8*8+4: printf("KING "); break;  
  
    default: printf("NOKING");  
  
}  
  
printf("PALACE");  
}
```

- AANT KING PALACE
- BKING PALACE
- CANT PALACE
- DCompiler error for using expressions

Correct Answer :C

Explanation

ASCII value of 'A' is 65. So (64+1) matches A. Also expression leading to Integers can be used as CASE Constants.

#757 [Explained](#) [Report](#) [Bookmark](#)

What is the size of the below C structure in TurboC?

```
int main()

{

    struct books{

        int pages;

        char str[4];

    }b;

    printf("%d",sizeof(b)) ;

    return 0;

}
```

- A5
- B6
- C7
- D8

Correct Answer :B

Explanation

2+4= 6 bytes.

#758 **Not Explained** Report Bookmark

In a nested structure definition, with country.state.district statement, member state is actually present in the structure.? (COUNTY, STATE, DISTRICT structures)

- A district
- B state
- C country
- D None of the above

Correct Answer :C

No Explanation Available

#759 **Explained** Report Bookmark

What is the output of c program with structures.?

```
int main()

{
    struct car

    {int color;};

    struct garage
```

```
{  
  
    struct car mycar[10];  
  
}gar;  
  
struct car c1={5};  
  
gar.mycar[0]=c1;  
  
printf("%d",gar.mycar[0]);  
  
return 0;  
}
```

- A NULL
- B 0
- C 5
- D Compiler error

Correct Answer :C

Explanation

It is an example of nested structures.

#760 [Explained](#) [Report](#) [Bookmark](#)

What is actually passed if you pass a structure variable to a function.?

- ACopy of structure variable
- BReference of structure variable
- CStarting address of structure variable
- DEnding address of structure variable

Correct Answer :B

Explanation

Yes. If you pass a structure variable by value without & operator, only a copy of the variable is passed. So changes made within that function do not reflect in the original variable.

#761 Explained Report Bookmark

What is the output of a C program with functions.?

```
void show();
```

```
void main()
```

```
{
```

```
    show();
```

```
    printf("RAINBOW ");
    return;
}

void show()
{
    printf("COLOURS ");
}
```

- A RAINBOW COLOURS
- B COLOURS RAINBOW
- C COLOURS
- D Compiler error

Correct Answer :B

Explanation

VOID functions should not return anything. RETURN; is returning nothing. 1. First void main() return; nothing. Still it is valid. 2. Second void show() function is NO RETURN statement. It is also valid.

#762 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with functions and pointers.?

```
int myshow(int *);

void main()
{
    int a=10;

    myshow(&a);

}

int myshow(int *k)
{
    printf("Received %d, ", *k);

}



- A Received RANDOM Number,
- B Received 10,
- C Received 01,
- D Compiler error

```

Correct Answer :B

Explanation

It is called Passing a variable by reference. You are passing &a instead of a. Address of a or &a is received as int *k. Observe the function prototype declaration before main(), int myshow(int *).

#763 [Explained](#) [Report](#) [Bookmark](#)

Choose correct statements about C Language Pass By Value.

- **A**
Pass By Value copies the variable value in one more memory location.
- **B**
Pass By Value does not use Pointers.
- **C**
Pass By Value protects your source or original variables from changes in outside functions or called functions.
- **D**
All the above

Correct Answer :D

Explanation

Point 1 & 3: Pass by value means that a copy of the actual parameter's value is made in memory, i.e. the caller and callee have two independent variables with the same value. If the callee modifies the parameter value, the effect is not visible to the caller

Point 2 : When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C.

#764 [Explained](#) [Report](#) [Bookmark](#)

Far pointer can access _____

- A Single memory location
- B No memory location
- C All memory location
- D First and Last Memory Address

Correct Answer :C

Explanation

A far pointer can access any segment of a RAM memory including all the 16 segments.

A far pointer consumes 32 kilobytes (4 bytes) of memory in your CPU. The first 16 bits stores segment value and the second 16 bits stores offset value. They may reference up to 1024 KB or 1088 KB of memory.

#765 [Explained](#) [Report](#) [Bookmark](#)

A pointer pointing to a memory location of the variable even after deletion of the variable is known as _____

- A far pointer

- **B**
dangling pointer
- **C**
void pointer
- **D**
null pointer

Correct Answer :B

Explanation

In short pointer pointing to non-existing memory location is called dangling pointer. if any pointer is pointing the memory address of any variable but after some variable has deleted from that memory location and still that pointer is pointing that location then it is called dangling pointer...

#766 [Explained](#) [Report](#) [Bookmark](#)

An uninitialized pointer in C is called _____

- **A**
Constructor
- **B**
dangling pointer
- **C**
Wild Pointer
- **D**
Destructor

Correct Answer :C

Explanation

Uninitialized pointers are called as wild pointers in C which points to arbitrary (random) memory location. This wild pointer may lead a program to behave wrongly or to crash.

#767 [Explained](#) [Report](#) [Bookmark](#)

A pointer that is pointing to **NOTHING** is called _____

- **A**
VOID Pointer
- **B**
DANGLING Pointer
- **C**
NULL Pointer
- **D**
WILD Pointer

Correct Answer :C

Explanation

NULL Pointer is a pointer which is pointing to nothing. In case, if we don't have address to be assigned to a pointer, then we can simply use **NULL**.

#768 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C Program.?

```
int main()
```

```
{
```

```
    if( 4 > 5 )
```

```
{  
  
    printf("Hurray..\n");  
  
}  
  
printf("Yes");  
  
  
  
  
return 0;  
  
}
```

- A Yes
- B Hurray.. Yes
- C Hurray..Yes
- D Compiler error

Correct Answer :A

Explanation

if condition fails. So control will not enter Hurray printf statement.

#769 [Explained](#) [Report](#) [Bookmark](#)

A function which calls itself is called a ___ function.

- A Self Function
- B Auto Function
- C Recursive Function

- D
Static Function

Correct Answer :C

Explanation

Recursion is a process in which a function calls itself as a subroutine. This allows the function to be repeated several times, since it calls itself during its execution. Functions that incorporate recursion are called recursive functions.

#770 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with functions.?

```
void show();

int main()
{
    show();
    printf("ARGENTINA ");
    return 0;
}
```

```
void show()  
  
{  
  
    printf("AFRICA ");  
  
}
```

- AARGENTINA AFRICA
- BAFRICA ARGENTINA
- CARGENTINA
- DCompiler error

Correct Answer :B

Explanation

First show() function is called. So it prints AFRICA first.

#771 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with Switch Statement.?

```
int main()  
  
{
```

```
    int a=5;
```

```
switch(a)

{

    case 0: printf("0 ");

    case 3: printf("3 ");

    case 5: printf("5 ");

    default: printf("RABBIT ");

}
```

```
a=10;

switch(a)

{

    case 0: printf("0 ");

    case 3: printf("3 ");

    case 5: printf("5 ");

    default: printf("RABBIT "); break;

}
```

```
    return 0;  
}
```

- A
5 RABBIT
- B
5 5 RABBIT 5 5 RABBIT
- C
0 3 5 RABBIT
- D
5 5 RABBIT RABBIT

Correct Answer :D

Explanation

Absence of break; after case statement causes control to go to next case automatically.

So after matching 5 with `a==5`, program prints 5 and control falls down the ladder without checking case again printing everything below it.

So Switch checks only once and falls down.

so first switch it will print 5 RABBIT then in second switch it print RABBIT as default case executed

#772 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement.?

```
int main()
```

```
{  
  
    int a=3;  
  
    switch(a)  
  
    {  
  
        case 2: printf("ZERO "); break;  
  
        case default: printf("RABBIT ");  
  
    }  
  
}  
  
}
```

- A RABBIT
- B ZERO RABBIT
- C No output
- D Compiler error

Correct Answer :D

Explanation

Notice that it is "default" not "case default".

#773 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```
int main()
{
    int a=3;

    switch(a)
    {
        }

    printf("MySwitch");
}

• A MySwitch
• B No Output
• C Compiler Error
• D None of the above
```

Correct Answer :D

Explanation

Yes. It is okay to omit CASE: statements inside SWITCH construct or block.

#774 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with switch statement or block.?

```
int main()
{
    int a;

    switch(a);
    {
        printf("DEER ");
    }

    printf("LION");
}
```

- A LION
- B DEER LION

- **CCompiler error**
- **DNone of the above**

Correct Answer :B

Explanation

Notice a semicolon at the end of switch(a);. So, printf DEER is out of SWITCH. `switch(a) { ; } { printf("DEER "); }`

#775 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```
int main()

{
    static int a=5;

    switch(a)

    {
        case 0: printf("ZERO ");break;

        case 5: printf("FIVE ");break;

        case 10: printf("DEER ");

    }
}
```

```
    printf("LION");  
}  
  
• A ZERO FIVE DEER LION  
• B FIVE DEER LION  
• C FIVE LION  
• D Compiler error
```

Correct Answer :C

Explanation

After matching 5, FIVE will be printed. BREAK causes control to exit SWITCH immediately. Also, using a STATIC variable is also allowed.

#776 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with switch statement or block.?

```
int main()  
  
{  
  
    char code='K';  
  
    switch(code)  
  
    {
```

```

        case 'A': printf("ANT ");break;

        case 'K': printf("KING "); break;

    default: printf("NOKING");

}

printf("PALACE");

}

```

- A KING PALACE
- B KING NOTHING PALACE
- C ANT KING PALACE
- D Compiler error for using Non Integers as CASE constants.

Correct Answer :A

Explanation

**Any character is replaced by ASCII code or number by the compiler.
So it is allowed to use CHARACTER constants for CASE constants.**

#777 [Explained](#) [Report](#) [Bookmark](#)

Which loop is most suitable to first perform the operation and then test the condition?

- Afor loop
- Bwhile loop
- Cdo-while loop
- Dnone of the mentioned

Correct Answer :C

Explanation

do-while loop

#778 [Explained](#) [Report](#) [Bookmark](#)

What will be the Output ?

```
#include

int main()

{
    int a[][] = {1, 2, 3, 4, 5, 6};

    int (*ptr)[3] = a;

    printf("%d %d ", (*ptr)[1], (*ptr)[2]);
    ++ptr;
    printf("%d %dn", (*ptr)[1], (*ptr)[2]);

    return 0;
}
```

- A 2 3 5 6
- B 2 3 4 5
- C 4 5 0 0
- D None

Correct Answer :A

Explanation

2 3 5 6n

#779 [Explained](#) [Report](#) [Bookmark](#)

What is the way to initialize array ?

- A int num[6] = { 2, 4, 12, 5, 45, 5 };
- B int n{} = { 2, 4, 12, 5, 45, 5 };
- C nt n{6} = { 2, 4, 12 };
- D int n(6) = { 2, 4, 12, 5, 45, 5 };

Correct Answer :A

Explanation

option (B), (C) and (D) are incorrect because array declaration syntax is wrong. Only square brackets([]) must be used for declaring an array.

#780 [Explained](#) [Report](#) [Bookmark](#)

What Will Be the Output ?

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

```

#include <stdlib.h>

int main(void)
{
    int i;
    int *ptr = (int *) malloc(5 * sizeof(int));
    for (i=0; i<5; i++)
        *(ptr + i) = i;
    printf("%d ", *ptr++);
    printf("%d ", (*ptr)++;
    printf("%d ", *ptr);
    printf("%d ", *++ptr);
    printf("%d ", ++*ptr);
}

```

- A Compiler Error
- B 0 1 2 2 3
- C 0 1 2 3 4
- D 1 2 3 4 5

Correct Answer :B

Explanation

The important things to remember for handling such questions are 1) Prefix ++ and * operators have same precedence and right to left associativity. 2) Postfix ++ has higher precedence than the above two mentioned operators and associativity is from left to right. We can apply the above two rules to guess all *ptr++ is treated as *(ptr++) *++ptr is treated as *(++ptr) ++*ptr is treated as ++(*ptr)

#781 [Explained](#) [Report](#) [Bookmark](#)

What will be the output ?

```
#include <stdio.h>

int main()

{
    int a[5] = {1,2,3,4,5};

    int *ptr = (int*)(&a+1);

    printf("%d %d", *(a+1), *(ptr-1));

    return 0;
}
```

- A 2 5
- B Garbage Value
- C Compiler Error
- D Segmentation Fault

Correct Answer :A

Explanation

The program prints “2 5”. Since compilers convert array operations in pointers before accessing the array elements, ($a+1$) points to 2. The expression ($\&a + 1$) is actually an address just after end of array (after address of 5) because $\&a$ contains address of an item of size $5 * \text{integer_size}$ and when we do ($\&a + 1$) the pointer is incremented by $5 * \text{integer_size}$. ptr is type-casted to int^* so when we do $\text{ptr} - 1$, we get address of 5

#782 Explained Report Bookmark

Let x be an array. which of the following operations are illegal ?

1. $++x$
 2. $x+1$
 3. $x++$
 4. x^2
- A1 & 2
 - B1, 2 & 3
 - C2 & 3
 - D1,3 &4

Correct Answer :D

Explanation

`int x[10];` * x will store the base address of array. * Statement 1, 3 and 4 is invalid. Statement 1 and 3 : $++x$ and $x++$ are throwing an error while compile (lvalue required as increment operand) Since, x is storing in the address of the array which is static value which cannot be change by the operand. Statement 4 : x^2 is also throw an error while compile (invalid operands to binary * (have 'int*' and 'int'))

Statement 2 : `x+1` is throw a warning: assignment makes integer from pointer without a cast [enabled by default]

#783 [Explained](#) [Report](#) [Bookmark](#)

An operation with only one operand is called unary operation.

- A Yes
- B An operation with two operand is called unary operation
- C An operation with unlimited operand is called unary operation
- D None of the above

Correct Answer :A

Explanation

Unary operator acts on single expression.

#784 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
int x = 5;
```

```
int* f()
```

```
{
```

```
    return &x;
```

```
}

main()

{

    *f() = 10;

    printf("%d", x);

}
```

- A Compile error
- B Runtime error
- C 5
- D 10

Correct Answer :D

Explanation

The returned address is global variables and 10 being stored in it.
Therefore x is 10.

#785 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
void f()  
  
{  
  
    static int i = 3;  
  
    printf("%d ", i);  
  
    if(--i) f();  
  
}  
  
main()  
  
{  
  
    f();  
  
}
```

- A 3 2 1 0
- B 3 2 1
- C 3 3 3
- D Compile error

Correct Answer :B

Explanation

As the static variable retains its value from the function calls, the recursion happens thrice.

#786 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
main()
```

```
{
```

```
    int a[3] = {2,1};
```

```
    printf("%d", a[a[1]]);
```

```
}
```

- A-0
- B-1
- C-2
- D-3

Correct Answer :B

Explanation

1, The inner indirection evaluates to 1, and the value at index 1 for outer indirection is 1.

#787 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

```
#include<stdio.h>

void swap(int m, int n)
{
    int x = m;
    m = n;
    n = x;
}

main()
{
    int x=5, y=3;
```

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

- A 3 5
- B 5 3
- C 5 5
- D Compile error

Correct Answer :B

Explanation

5 3, call by value mechanism can't alter actual arguments.

#788 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the following program?

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

```
void main()
```

```
{
```

```
char *s = "C++";  
  
printf("%s ", s);  
  
s++;  
  
printf("%s", s);  
  
}
```

- A C++ C++
- B C++ ++
- C ++ ++
- D Compile error

Correct Answer :B

Explanation

After s++, s points the string “++”.

#789 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```
int main()  
{
```

```

switch(24.5)

{

    case 24.5: printf("SILVER ") ;break;

    case 25.0: printf("GOLD ") ; break;

    default: printf("TIN ");

}

printf("COPPER");

}

```

- A SILVER COPPER
- B TIN COPPER
- C COPPER
- D Compiler error

Correct Answer :D

Explanation

You can not use float, double or Strings inside Switch or Switch CASE.

#790 Explained Report Bookmark

In the given below statement, what does the “arr” indicate?

```
char *arr[30];
```

- **A**arr is a array of function
- **B**arr is a array of 30 characters
- **C**arr is a pointer to an array
- **D**arr is a array of 30 character pointers

Correct Answer :D

Explanation

square parenthesis signify as array at declaration and type is char*, so array of character pointers.

#791 [Explained](#) [Report](#) [Bookmark](#)

Correct way of commenting a single line is.?

- **A***printf("Hello C.."); printf("How are you.");
- **B**//printf("Hello C.."); printf("How are you.");
- **C**/*printf("Hello C.."); printf("How are you.");*/
- **D**printf("Hello C..");/ printf("How are you.");

Correct Answer :B

Explanation

Answer C comments two lines with Multi-Line comment or BLOCK Comment characters /* */ Only // is a Single Line Commenting characters.

#792 Explained Report Bookmark

Choose the correct statement about a C Switch Construct.

- A default case is optional inside switch.
- B break; causes the control to exit the switch immediately and avoid fall down to other CASE statements.
- C You can not use duplicate CASE Constants inside a Switch construct.
- D All the above.

Correct Answer :D

Explanation

Option a: The default case is an optional one. Whenever the value of test-expression is not matched with any of the cases inside the switch, then the default will be executed. ... Once the switch is executed the control will go to the statement-x, and the execution of a program will continue

Option b: break statement to get the output from that particular case only. Note that break need not be written after the default statement since the control comes out of the switch loop anyway.

Option c: The values of CASE constant the case keyword can be an integer or character. But all these constants must be different from each other.

#793 Explained Report Bookmark

An Identifier may contain.?

- **A**
Letters a-z, A-Z in Basic character set. Unicode alphabet
characters other languages
- **B**
Underscore _ symbol
- **C**
Numbers 0 to 9 Unicode Numbers in other languages
- **D**
All the above

Correct Answer :D

Explanation

Following rules must be followed for identifiers:

1. The first character must always be an alphabet or an underscore.
2. It should be formed using only letters, numbers, or underscore.
3. A keyword cannot be used as an identifier.
4. It should not contain any whitespace character.
5. The name must be meaningful.

#794 [Explained](#) [Report](#) [Bookmark](#)

What is an Identifier in C Language.?

- **A**Name of a Function or Variable
- **B**Name of a Macros
- **C**Name of Structure or Union
- **D**All the above.

Correct Answer :D

Explanation

int age=25; //here age is an Identifier

#795 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with structures.?

```
int main()
{
    struct paint{
        int type;
        int color;
    }p1, p2;
    p1.type=1;
    p1.color=5;
    if(sizeof(p1)==sizeof(p2))
    {
        printf("SAME");
    }
}
```

```
    else

    {

        printf("DIFFERENT") ;

    }

    return 0;

}
```

- A SAME
- B DIFFERENT
- C Compiler error
- D None of the above

Correct Answer :A

Explanation

Yes. Before and after initialization, size of a structure variable does not change.

#796 [Explained](#) [Report](#) [Bookmark](#)

Number of Keywords present in C Language are .?

- A 32
- B 34
- C 62
- D 64

Correct Answer :A

Explanation

Only 32 Keywords originally. Compilers are individual companies can include and use extra keywords if required. Such keywords should precede with __ (two Underscore symbols before names). eg.
__mykeyword

#797 [Explained](#) [Report](#) [Bookmark](#)

What is the size of a C structure.?

- A
C structure is always 128 bytes.
- B
Size of C structure is the size of largest element.
- C
Size of C structure is the total bytes of all elements of structure.
- D
None of the above

Correct Answer :C

Explanation

Explanation:

- Structure is a user-defined datatype in C language which allows us to combine data of different types together.
- The Size of C structure is the total bytes of all elements of structure.
- Individually calculate the sizes of each member of a structure and make a total to get Full size of a structure.

#798 [Explained](#) [Report](#) [Bookmark](#)

What is a structure in C language.?

- **A**
A structure is a collection of elements that can be of different data type.
- **B**
A structure is a collection of elements that can be of same data type.
- **C**
Elements of a structure are called members.
- **D**
All the above

Correct Answer :D

Explanation

Structure is a user-defined datatype in C language which allows us to combine data of different types together.

Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only

```
struct insurance { int age; char name[20]; }
```

#799 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

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

```
int main()
```

```
{
```

```
int x=12, y=7, z;  
  
z = x!=4 || y == 2;  
  
printf("z=%d\n", z);  
  
return 0;  
  
}
```

- A z=0
- B z=1
- C z=4
- D z=2

Correct Answer :B

Explanation

Step 1: int x=12, y=7, z; here variable x, y and z are declared as an integer and variable x and y are initialized to 12, 7 respectively. Step 2: z = x!=4 || y == 2; becomes z = 12!=4 || 7 == 2; then z = (condition true) || (condition false); Hence it returns 1. So the value of z=1. Step 3: printf("z=%d\n", z); Hence the output of the program is "z=1".

#800 [Explained](#) [Report](#) [Bookmark](#)

Choose a right statement.

- A int myage = 10; int my_age = 10;
- B nt myage = 10; int my,age = 10;
- C int myage = 10; int my age = 10;
- D All are right

Correct Answer :A

Explanation

Only Underscore (_) symbol is allowed in a variable name i.e identifier name. Space, Comma and other special characters are not allowed.

#801 [Explained](#) [Report](#) [Bookmark](#)

A Variable of a particular type can hold only a constant of the same type. Choose right answer.

- A TRUE
- B FALSE
- C depends on the place the variable is declared.
- D None of the above.

Correct Answer :A

Explanation

An int can hold only Integer constant. A float can hold only Real Number constants. A char can hold only Character constants.

#802 [Explained](#) [Report](#) [Bookmark](#)

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

```
int main()
```

```
{
```

```
enum status { pass, fail, atkt};
```

```
enum status stud1, stud2, stud3;
```

```
stud1 = pass;  
stud2 = atkt;  
stud3 = fail;  
  
printf("%d, %d, %d\n", stud1, stud2, stud3);  
  
return 0;  
  
}
```

- A 0, 1, 2
- B 1, 2, 3
- C 0, 2, 1
- D 1, 3, 2

Correct Answer :C

Explanation

enum takes the format like {0,1,2..) so pass=0, fail=1, atkt=2 stud1 = pass (value is 0) stud2 = atkt (value is 2) stud3 = fail (value is 1) Hence it prints 0, 2, 1

#803 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program.?

```
int main()  
  
{  
  
    struct ship
```

```
{  
  
    int size;  
  
    char color[10];  
  
}boat1, boat2;  
  
boat1.size=10;  
  
boat2 = boat1;  
  
printf("boat2=%d",boat2.size);  
  
return 0;  
}
```

- A boat2=0
- B boat2=-1
- C boat2=10
- D Compiler error

Correct Answer :C

Explanation

Yes, it is allowed to assign one structure variables. boat2=boat1. Remember, boat1 and boat2 have different memory locations.

#804 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#include<stdio.h>

int main()

{

char str[20] = "Hello";

char *const p=str;

*p='M';

printf("%s\n", str);

return 0;

}
```

- A
Mello
- B
Hello
- C
HMello
- D
MHello

Correct Answer :A

Explanation

[H][e][l][l][o][w]

10 11 12 13 14 15 (Assume these are address of memory)

here,str=10

char*const p=str (here,p will store 10 that is constant)

so,now *p indicate "H" that has replaced by "M"

note: Base address also is a pointer & constant

#805 [Explained](#) [Report](#) [Bookmark](#)

Choose a correct statement about C structures.

- A Structure elements can be initialized at the time of declaration.
- B Structure members can not be initialized at the time of declaration
- C Only integer members of structure can be initialized at the time of declaration
- D None of the above

Correct Answer :B

Explanation

struct book { int SNO=10; //not allowed };

#806 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about the program below?

```
#include<stdio.h>

int main()

{
    int size, i;

    scanf("%d", &size);

    int arr[size];

    for(i=1; i<=size; i++)

    {
        scanf("%d", arr[i]);

        printf("%d", arr[i]);
    }

    return 0;
}
```

- A The code is erroneous since the subscript for array used in for loop is in the range 1 to size.
- B The code is erroneous since the values of array are getting scanned through the loop.
- C The code is erroneous since the statement declaring array is invalid.
- D The code is correct and runs successfully.

Correct Answer :C

Explanation

The statement `int arr[size];` produces an error, because we cannot initialize the size of array dynamically. Constant expression is required here. Example: `int arr[10];` One more point is there, that is, usually declaration is not allowed after calling any function in a current block of code. In the given program the declaration `int arr[10];` is placed after a function call `scanf()`.

#807 [Explained](#) [Report](#) [Bookmark](#)

Choose a incorrect statement about C structure.?

```
int main()
{
    struct ship
```

- ```
 } ;

 return 0;
}

 - A It is wrong to define an empty structure
 - B Member variables can be added to a structure even after its first definition.
 - C There is no use of defining an empty structure
 - D None of the above
```

Correct Answer :B

## Explanation

After structure declaration you can't add the member variables in structure

#808 Explained Report Bookmark

What is the output of C program with structures.?

```
int main()
{
 structure hotel
```

```

{

 int items;

 char name[10];

}a;

strcpy(a.name, "TAJ");

a.items=10;

printf("%s", a.name);

return 0;

}

```

- A **TAJ**
- B **Empty string**
- C **Compiler error**
- D **None of the above**

**Correct Answer :A**

## Explanation

**Keyword used to declare a structure is STRUCT not structURE in lowercase i.e struct.**

#809 [Explained](#) [Report](#) [Bookmark](#)

**How many bytes are occupied by *near*, *far* and *huge* pointers (DOS)?**

- A **A**near=2 far=4 huge=4
- B **B**near=4 far=8 huge=8
- C **C**near=2 far=4 huge=8
- D **D**near=4 far=4 huge=8

Correct Answer :A

## Explanation

**near=2, far=4 and huge=4** pointers exist only under DOS. Under windows and Linux every pointers is 4 bytes long.

#810 Explained Report Bookmark

What is the output of the below code snippet?

```
#include

main()
{
 for()printf("InfoBeam");
}
```

- A **A**Infinte loop
- B **B**Prints “InfoBeam” once.
- C **C**No output
- D **D**Compile error

Correct Answer :D

## Explanation

Compiler error, semi colons need to appear though the expressions are optional for the 'for' loop.

#811 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program If the integer is 4bytes long?

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

```
int main()
```

```
{
```

```
int ***r, **q, *p, i=8;
```

```
p = &i;
```

```
q = &p;
```

```
r = &q;
```

```
printf("%d, %d, %d\n", *p, **q, ***r);
```

```
return 0;
```

```
}
```

- A  
8, 8, 8
- B  
4000, 4002, 4004

- C  
4000, 4004, 4008
- D  
4000, 4008, 4016

Correct Answer :A

## Explanation

1.  $p = \&i;$

Address of i is pointed by p pointer

1.  $q = \&p;$

Address of p is pointed by q pointer

1.  $r = \&q;$

Address of q is pointed by r pointer

1.  $*p$  : value at address pointed by p pointer is printed . here it is i which is 8
2.  $**q$  : value pointed by pointer which is pointing to i i.e. q pointing to p and p pointing to i
3.  $***r$  : value pointed by pointer pointed by another pointer which is pointing to a location i.e. r pointing to q which is pointing to p which is pointing to i=8
4. Thus 8,8,8 will be printed

#812 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#include<stdio.h>

void fun(int **p);

int main()

{

int a[3][4] = {1, 2, 3, 4, 4, 3, 2, 8, 7, 8, 9, 0};

int *ptr;

ptr = &a[0][0];

fun(&ptr);

return 0;

}

void fun(int **p)

{

printf("%d\n", **p);

}
```

- A1
- B2
- C3
- D4

Correct Answer :A

## Explanation

Step 1: int a[3][4] = {1, 2, 3, 4, 4, 3, 2, 8, 7, 8, 9, 0}; The variable a is declared as an multidimensional integer array with size of 3 rows 4 columns. Step 2: int \*ptr; The \*ptr is a integer pointer variable. Step 3: ptr = &a[0][0]; Here we are assigning the base address of the array a to the pointer variable \*ptr. Step 4: fun(&ptr); Now, the &ptr contains the base address of array a. Step 4: Inside the function fun(&ptr); The printf("%d\n", \*\*p); prints the value '1'. because the \*p contains the base address or the first element memory address of the array a (ie. a[0]) \*\*p contains the value of \*p memory location (ie. a[0]=1). Hence the output of the program is '1'

#813 [Explained](#) [Report](#) [Bookmark](#)

Is there any difference between following declarations?

1 : extern int fun();

2 : int fun();

- A Both are identical
- B No difference, except extern int fun(); is probably in another file
- C int fun(); is overridden with extern int fun();
- D None of these

Correct Answer :B

## Explanation

extern int fun(); declaration in C is to indicate the existence of a global function and it is defined externally to the current module or in

another file. int fun(); declaration in C is to indicate the existence of a function inside the current module or in the same file.

#814 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int main()

{
 int i=-3, j=2, k=0, m;

 m = ++i && ++j || ++k;

 printf("%d, %d, %d, %d\n", i, j, k, m);

 return 0;
}
```

- A  
1, 2, 0, 1
- B  
-3, 2, 0, 1
- C  
-2, 3, 0, 1

- D  
2, 3, 1, 1

Correct Answer :C

## Explanation

**Step 1:** *int i=-3, j=2, k=0, m;* here variable *i, j, k, m* are declared as an integer type and variable *i, j, k* are initialized to -3, 2, 0 respectively.

**Step 2:** *m = ++i && ++j // ++k;*

becomes *m = -2 && 3 &// 0 ;*

becomes *m = TRUE && TRUE;* Hence this statement becomes TRUE. So it returns '1'(one). Hence m=1.

**Step 3:** *printf("%d, %d, %d, %d\n", i, j, k, m);* In the previous step the value of i,j,( k i not incremented as first part i true and we have or condition here )

Hence the output is "-2, 3, 0, 1".

#815 [Explained](#) [Report](#) [Bookmark](#)

Point out the error/warning in the program?

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

```

int main()

{

 unsigned char ch;

 FILE *fp;

 fp=fopen("trial", "r");

 while((ch = getc(fp)) !=EOF)

 printf("%c", ch);

 fclose(fp);

 return 0;

}

```

- A**Error: in unsigned char declaration**
- B**Error: while statement**
- C**No error**
- D**It prints all characters in file "trial"**

**Correct Answer :A**

## Explanation

**Here, EOF is -1. As 'ch' is declared as unsigned char it cannot deal with any negative value.**

#816 [Explained](#) [Report](#) [Bookmark](#)

**What will be the output of the program (sample.c) given below if it is executed from the command line (Turbo C in DOS)?**

**cmd> sample 1 2 3**

```
/* sample.c */

#include <stdio.h>

int main(int argc, char *argv[])
{
 int j;

 j = argv[1] + argv[2] + argv[3];

 printf("%d", j);

 return 0;
}
```

- A 6
- B sample 6
- C Error
- D Garbage value

Correct Answer :C

## Explanation

Here argv[1], argv[2] and argv[3] are string type. We have to convert the string to integer type before perform arithmetic operation.

Example: j = atoi(argv[1]) + atoi(argv[2]) + atoi(argv[3]);

#817 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program?

```
#include<stdio.h>

int addmult(int ii, int jj)

{
 int kk, ll;

 kk = ii + jj;

 ll = ii * jj;

 return (kk, ll);
}
```

```

int main()

{

 int i=3, j=4, k, l;

 k = addmult(i, j);

 l = addmult(i, j);

 printf("%d, %d\n", k, l);

 return 0;

}

```

- A12, 12
- B7, 7
- C7, 12
- D12, 7

Correct Answer :A

## Explanation

**Step 1:** int i=3, j=4, k, l; The variables i, j, k, l are declared as an integer type and variable i, j are initialized to 3, 4 respectively. The function addmult(i, j); accept 2 integer parameters.

**Step 2:** k = addmult(i, j); becomes k = addmult(3, 4) In the function addmult(). The variable kk, ll are declared as an integer type int kk, ll; kk = ii + jj; becomes kk = 3 + 4 Now the kk value is '7'. ll = ii \* jj; becomes ll = 3 \* 4 Now the ll value is '12'. return (kk, ll); It returns the value of variable ll only. The value 12 is stored in variable 'k'.

**Step 3:** l = addmult(i, j); becomes l =

**addmult(3, 4) kk = ii + jj; becomes kk = 3 + 4 Now the kk value is '7'. ll = ii \* jj; becomes ll = 3 \* 4 Now the ll value is '12'. return (kk, ll); It returns the value of variable ll only. The value 12 is stored in variable 'l'. Step 4: printf("%d, %d\n", k, l); It prints the value of k and l Hence the output is "12, 12".**

#818 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#include<stdio.h>

int *check(static int, static int);

int main()

{

 int *c;

 c = check(10, 20);

 printf("%d\n", c);

 return 0;

}

int *check(static int i, static int j)
```

```

{
 int *p, *q;

 p = &i;

 q = &j;

 if(i >= 45)

 return (p);

 else

 return (q);
}

```

- **A**  
10
- **B**  
20
- **C**  
**Error: Non portable pointer conversion**
- **D**  
**Error: cannot use static for function parameters**

Correct Answer :D

## Explanation

The static keyword means that a variable may have only and exactly one instance in its scope, and that instance is invisible out of its scope ( cannot be used in any other function than where it is defined).

Neither of these requirements make sense for a function argument: it may be called multiple times, at a different memory address, and as it's meant for communication, it has to be visible to the outer world.

typical handling of function parameter passing uses stack. Suppose if we use static variables for function parameters, then it would mean that they shouldn't be passed around in the stack and will have to be referred from the sticky location. This would be a special case and the implementation to handle this will have to be different than the handling for the normal function calls. Some compilers might allow this(mostly embedded ones or the once with extensions).

#819 [Explained](#) [Report](#) [Bookmark](#)

If the size of integer is 4bytes, What will be the output of the program?

```
#include<stdio.h>

int main()
{
 int arr[] = {12, 13, 14, 15, 16};

 printf("%d, %d, %d\n", sizeof(arr), sizeof(*arr),
sizeof(arr[0]));

 return 0;
}
```

- A  
10, 2, 4
- B  
20, 4, 4
- C  
16, 2, 2
- D  
20, 2, 2

Correct Answer :B

## Explanation

1. **sizeof(arr):** array size is printed . here total elements are 5 each of 4 Byte thus total memory consumed will be  $5*4=20$
2. **sizeof(\*arr):** size of first element will be printed which is 4Byte since it's integer since there is no pointer allocated to it
3. **sizeof(arr[0]):** size of 0th element is printed which is 4Byte since its integer

#820 [Explained](#) [Report](#) [Bookmark](#)

In the following program add a statement in the function fact() such that the factorial gets stored in j.

```
#include<stdio.h>

void fact(int*) ;

int main()
```

```
{\n\n int i=5;\n\n fact(&i);\n\n printf("%d\\n", i);\n\n return 0;\n}\n\nvoid fact(int *j)\n{\n static int s=1;\n\n if(*j!=0)\n {\n s = s**j;\n\n *j = *j-1;\n\n fact(j);\n\n /* Add a statement here */\n }\n}
```

- A  
j=s;
- B  
\*j=s;
- C  
\*j=&s;
- D  
&j=s;

Correct Answer :B

## Explanation

\*j=s will be the right answer here. Since the program is if factorial, 'i' is supposed to hold the factorial too. Thus we are storing the value of s which is factorial in \*j

#821 Explained Report Bookmark

What will be the output of the program if the array begins at address 65486?

```
#include<stdio.h>

int main()

{
 int arr[] = {12, 14, 15, 23, 45};

 printf("%u, %u\n", arr, &arr);
```

```
 return 0;
}

• A
 65486, 65488
• B
 65486, 65486
• C
 65486, 65490
• D
 65486, 65487
```

Correct Answer :B

## Explanation

**Step 1:** *int arr[] = {12, 14, 15, 23, 45};* The variable **arr** is declared as an integer array and initialized.

**Step 2:** *printf("%u, %u\n", arr, &arr);* Here,

The base address of the array is 65486.

=> **arr, &arr** is pointing to the base address of the array **arr**.

Hence the output of the program is 65486, 65486

#822Explained Report Bookmark

What will be the output of the program ?

```
#include<stdio.h>

int main()

{
 int i;

 char a[] = "\0";

 if(sprintf("%s", a))

 printf("The string is empty\n");

 else

 printf("The string is not empty\n");

 return 0;
}
```

- A The string is empty
- B The string is not empty
- C No output
- D 0

Correct Answer :B

# Explanation

The function printf() returns the number of characters printed on the console. Step 1: char a[] = "\0"; The variable a is declared as an array of characters and it initialized with "\0". It denotes that the string is empty. Step 2: if(strcmp("%s", a)) The printf() statement does not print anything, so it returns '0'(zero). Hence the if condition is failed. In the else part it prints "The string is not empty".

#823 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following program in 16 bit platform assuming that 1022 is memory address of the string "Hello1" (in Turbo C under DOS) ?

```
#include<stdio.h>

int main()

{
 printf("%u %s\n", &"Hello1", &"Hello2");

 return 0;
}
```

- A1022 Hello2

- **B**Hello1 1022
- **C**Hello1 Hello2
- **D**1022 1022

Correct Answer :A

## Explanation

In `printf("%u %s\n", &"Hello", &"Hello");`. The `%u` format specifier tells the compiler to print the memory address of the "Hello1". The `%s` format specifier tells the compiler to print the string "Hello2". Hence the output of the program is "1022 Hello2".

#824 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about the program?

```
#include<stdio.h>

int main()

{

 unsigned int num;

 int i;

 scanf("%u", &num);
```

```

for(i=0; i<16; i++)

{
 printf("%d", (num<<i & 1<<15)?1:0);

}

return 0;
}

```

- **A**lt prints all even bits from num
- **B**lt prints all odd bits from num
- **C**lt prints binary equivalent num
- **D**Error

**Correct Answer :C**

## Explanation

If we give input 4, it will print 00000000 00000100 ; If we give input 3, it will print 00000000 00000011 ; If we give input 511, it will print 00000001 11111111 ;

#825 [Explained](#) [Report](#) [Bookmark](#)

**Choose a right statement.**

```

int main()

{

```

```
float c = 3.5 + 4.5;

printf("%d", (int)c);

return 0;

}
```

- A8.0
- B8.0000000
- C7
- D8

Correct Answer :D

## Explanation

You are printing a float variable by type casting to int. So integer is printed. int c = 3.5 + 4.5 also holds and prints 8.

#826 [Explained](#) [Report](#) [Bookmark](#)

Choose a right statement.

```
int a = 5/2;
```

```
int b = 5.0/2;
```

```
int c = 5 / 2.0;
```

```
int d = 5.0/2.0;
```

- A a = 2, b = 2, c = 2, d= 2
- B a = 2, b = 2.0, c = 2, d= 2.0
- C a = 2, b = 2.5, c = 2.5, d= 2.5
- D a = 2.5, b = 2.5, c = 2.5, d= 2.5

Correct Answer :A

## Explanation

Irrespective of numbers after decimal point, an int variable holds only integer value i.e 2.

#827 [Explained](#) [Report](#) [Bookmark](#)

Can you use C Modulo Division operator % with float and int?

- A Only int variables = Okay
- B Only float variables = Okay
- C int or float combination = Okay
- D Numerator int variable, Denominator any variable = Okay

Correct Answer :A

## Explanation

Modulo Division operator % in C language can be used only with integer variables or constants.

#828 [Explained](#) [Report](#) [Bookmark](#)

What is the output of the C program with Modulo Division operator with - or Negative numbers.?

```
int main()

{
 int a = -25%-10;

 int b = -25%10;

 int c = 25%-10;

 printf("%d %d %d", a, b, c);

 return 0;
}

- A5 -5 -5
- B5 -5 5
- C-5 -5 5
- D5 5 5

```

Correct Answer :C

## Explanation

Sign of a modulo division operation is same as the sign of Numerator.  
So sign of 25 is taken always.

#829 [Explained](#) [Report](#) [Bookmark](#)

**With respect to following “for” loops in C, pick the best statement.  
Assume that there is a prior declaration of 'i' in all cases**

`for (i = 0; i < 10 ; i++) // (i)`

`for ( ; i < 10 ; i++) // (ii)`

`for (i = 0; ; i++) // (iii)`

`for (i = 0; i < 10 ; ) // (iv)`

`for ( ; ; ) // (v)`

- A Only (i) and (v) would compile successfully. Also (v) can be used as infinite loop.
- B Only (i) would compile successfully.
- C All would compile successfully but behavior of (ii), (iii) and (iv) would depend on compiler.
- D All would compile successfully

**Correct Answer :D**

## **Explanation**

In C, any of the 3 expressions of “for” loop can be empty. The exact behavior of the loop depends on the body of the loop as well. Basically, all of the 3 expressions of loop can be put inside the loop body. So as per C language standard, all of the above are valid for loops.

#830 [Explained](#) [Report](#) [Bookmark](#)

**What's going to happen when we compile and run the following C program snippet?**

```
#include "stdio.h"

int main()

{

int a = 10;

int b = 15;

printf("%d", (a+1), (b=a+2));

printf(" %d=", b);

return 0;

}
```

- A=11 15=
- B=11 12=
- Compiler Error due to (b=a+2) in the first printf()
- No compile error but output would be =11 X= where X would depend on compiler implementation.

**Correct Answer :B**

## **Explanation**

As per C standard C11, all the arguments of printf() are evaluated irrespective of whether they get printed or not. That's why (b=a+2) would also be evaluated and value of b would be 12 after first printf(). That's why correct answer is B.

#831 [Explained](#) [Report](#) [Bookmark](#)

For a given integer, which of the following operators can be used to “set” and “reset” a particular bit respectively?

- A| and &
- B&& and ||
- C& and |
- D|| and &&

Correct Answer :A

## Explanation

Bitwise operator | can be used to “set” a particular bit while bitwise operator & can be used to “reset” a particular bit. Please note that && and || are logical operators which evaluate their operands to logical TRUE or FALSE. It should be noted that bitwise operator & can be used for checking a particular bit also i.e. whether a bit is set or not. So correct answer it A.

#832 [Explained](#) [Report](#) [Bookmark](#)

**What's going to happen when we compile and run the following C program snippet?**

```
#include "stdio.h"

int main()

{
 int a = 10;

 printf("%d %d", (a+1));

 return 0;
}
```

- A=11 0=
- B=11 X= where X would depend on Compiler implementation

- **C**Undefined behaviour
- **D**Compiler Error due to missing argument for second %d

Correct Answer :C

## Explanation

In the context of printf() and fprintf(), as per C standard C11 clause 7.21.6.1, "If there are insufficient arguments for the format, the behavior is undefined. If the format is exhausted while arguments remain, the excess arguments are evaluated (as always) but are otherwise ignored." Some implementation can choose to print =10 0= while other implementations can choose to print =11 X=. That's why the output of above program varies depending on the compiler and platform. Hence, correct answer is C).

#833Explained Report Bookmark

With respect to following “for” loops in C, pick the best statement  
Assume that there is a prior declaration of 'i' in all cases

```
for (i < 10; i = 0 ; i++) // (i)

for (i < 10; i++ ; i = 0) // (ii)

for (i = 0; i < 10 ; i++) // (iii)

for (i = 0; i++ ; i < 10) // (iv)
```

```
for (i++; i = 0 ; i < 10) // (v)
```

```
for (i++; i < 0 ; i = 10) // (vi)
```

- A All the above “for” loops would compile successfully
- B All the above “for” loops would compile successfully. Except (iii), the behaviour of all the other “for” loops depend on compiler implementation.
- C Only (iii) would compile successfully
- D Only (iii) and (iv) would compile successfully.

Correct Answer :A

## Explanation

Basically, all of the “for” loops are valid i.e. . In the above examples, it doesn’t matter what expression has been put in which part of a “for” loop. The execution order of these expressions remain same irrespective of where they have been put i.e. "1st expression" followed by "2nd expression" followed by "body of the loop" followed by "3rd expression". But the exact behavior of each of the above "for" loop depends on the body of loop as well. In fact the following is also valid and work without any issue in C. `for(sprintf("1st") ; printf("2nd") ; printf("3rd")) { break; }`

#834 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```

int main()

{

 int k=25;

 switch(k)

 {

 case 24: printf("ROSE ") ;break;

 case 25: printf("JASMINE ") ; continue;

 default: printf("FLOWER ");

 }

 printf("GARDEN");

}

```

- A JASMINE GARDEN
- B JASMINE FLOWER GARDEN
- C FLOWER GARDEN
- D Compiler error

**Correct Answer :D**

## Explanation

You can not use CONTINUE; statement as SWITCH is not a LOOP like for, while and do while.

#835 [Explained](#) [Report](#) [Bookmark](#)

As per C language standard, which of the followings is/are not keyword(s)? Pick the best statement. auto make main sizeof elseif

- A None of the above is keywords in C.
- B make main elseif
- C make main
- D auto make

Correct Answer :B

## Explanation

"auto" is a storage specifier defined in C language. "sizeof" is unary operator defined in C language. Both of these are reserved words i.e. keywords as per C language standard. All the other 3 i.e. main make and elseif aren't keywords. In fact, you can use int main, make, elseif; in your C program.

#836 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C Program with switch statement or block.?

```
int main()

{

 int k=8;

 switch(k)

 {

 case 1==8: printf("ROSE ") ;break;

 case 1 && 2: printf("JASMINE ") ; break;

 default: printf("FLOWER ");

 }

 printf("GARDEN");

}
```

- A ROSE GARGEN
- B JASMINE GARDEN
- C FLOWER GARDEN
- D Compiler error

Correct Answer :C

## Explanation

**(1==8)** is false i.e 0. **(1&&2)** is true AND true i.e true i.e 1. So case allowed first ZERO 0 and then ONE 1. All integer constants are allowed for SWITCH Case Constants.

#837 [Explained](#) [Report](#) [Bookmark](#)

### Choose correct statements

- A Constant value does not change. A variable value can change according to needs.
- B A constant can change its values. A variable can have one constant value only.
- C There is no restriction on number of values for constants or variables.
- D Constants and Variables can not be used in a single main function.

Correct Answer :D

## Explanation

Constant value is always constant. Constant is also called Literal. Variable can have any number of arbitrary values and once value at any point of time. Variable is also called Identifier.

#838 [Explained](#) [Report](#) [Bookmark](#)

What's going to happen when we compile and run the following C program?

```
#include "stdio.h"
```

```
int main()
{
 int i = 1, j;
 for (; ;)
 {
 if (i)
 j = --i;
 if (j < 10)
 printf("InfoBeam ", j++);
 else
 break;
 }
 return 0;
}
```

- A Compile Error
- B No compile error but it will run into infinite loop printing InfoBeam
- C No compile error and it'll print InfoBeam 10 times
- D No compile error but it'll print InfoBeam 9 times

Correct Answer :C

## Explanation

Basically, even though the for loop doesn't have any of three expressions in parenthesis, the initialization, control and increment has been done in the body of the loop. So j would be initialized to 0 via first if. This if itself would be executed only once due to i--. Next if and else blocks are being used to check the value of j and existing the loop if j becomes 10. Please note that j is getting incremented in printf even though there's no format specifier in format string. That's why InfoBeam would be printed for j=0 to j=9 i.e. a total of 10 times.

#839 [Explained](#) [Report](#) [Bookmark](#)

Which of the following functions from “stdio.h” can be used in place of printf()?

- A fputs() with FILE stream as stdout
- B fprintf() with FILE stream as stdout
- C fwrite() with FILE stream as stdout.
- D All of the above three - a, b and c

Correct Answer :B

## Explanation

Though fputs() and fwrite() can accept FILE stream stdout and can output the given string but the input string wouldn't result in formatted (i.e. containing format specifiers) output. But fprintf() can be used for formatted output. That's why fprintf(stdout,"%d=",a); and printf("%d=",a); both are equivalent. The correct answer is B.

#840 [Explained](#) [Report](#) [Bookmark](#)

What's going to happen when we compile and run the following C program?

```
#include "stdio.h"

int main()

{
 int j = 0;

 for (; j < 10 ;)

 {
 if (j < 10)

 printf("ccat", j++);

 else

 continue;

 printf("ITS");

 }

 return 0;
}
```

- A  
Compile Error due to use of continue in for loop

- **B**  
No compile error but it will run into infinite loop printing InfoBeam
- **C**  
No compile error and it'll print ccat 10 times followed by ITS once.
- **D**  
No compile error and it'll print ccatITS 10 times

Correct Answer :D

## Explanation

Here we are looping the block from j=0 to j=9 i.e. j<10.

Since the j<10 condition is satisfied ccat is printed on screen. But as we are using the continue statement too “ITS” will be printed on screen too . It would be the same as writing “ITS” print statement inside the if block. Also removing the else block will result in the same output.

#841 [Explained](#) [Report](#) [Bookmark](#)

Comment on the following pointer declaration?

```
int *ptr, p;
```

- **A**ptr is a pointer to integer, p is not.
- **B**ptr and p, both are pointers to integer.
- **C**ptr is pointer to integer, p may or may not be.
- **D**null

Correct Answer :A

# Explanation

ptr and p both are not pointers to integer.

#842 [Explained](#) [Report](#) [Bookmark](#)

**What is the output of C Program with switch statement or block.?**

```
int main()
{
 int k=64;

 switch(k)
 {
 case k<64: printf("SHIP ") ;break;

 case k>=64: printf("BOAT ") ; break;

 default: printf("PETROL");

 }

 printf("CHILLY");
}
```

- ABOAT CHILLY
- BBOAT PETROL CHILLY
- CSHIP BOAT CHILLY
- DCompiler error

Correct Answer :D

## Explanation

You can not use comparison operations with CASE statements in SWITCH.

#843 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statement is correct for switch controlling expression?

- AOnly int can be used in “switch” control expression.
- BBoth int and char can be used in “switch” control expression
- CAll types i.e. int, char and float can be used in “switch” control expression.
- D“switch” control expression can be empty as well

Correct Answer :B

## Explanation

As per C standard, “The controlling expression of a switch statement shall have integer type.” Since char is promoted to integer in switch control expression, it’s allowed but float isn’t promoted. That’s why B is correct statement.

#844 [Explained](#) [Report](#) [Bookmark](#)

**Point out the error, if any in the program.**

```
#include<stdio.h>

int main()

{

 int a = 10, b;

 a >=5 ? b=100: b=200;

 printf("%d\n", b);

 return 0;

}
```

- A100
- B200
- CError: L value required for b
- DGarbage value

**Correct Answer :D**

## Explanation

**Variable b is not assigned. It should be like: b = a >= 5 ? 100 : 200;**

#845 [Explained](#) [Report](#) [Bookmark](#)

**How many times the while loop will get executed if a short int is 2 byte wide?**

```
#include

int main()

{

 int j=1;

 while(j <= 255)

 {

 printf("%c %d\n", j, j);

 j++;

 }

 return 0;

}
```

- A Infinite times
- B 255 times
- C 256 times
- D 254 times

**Correct Answer :B**

# Explanation

The while( $j \leq 255$ ) loop will get executed 255 times. The size short int(2 byte wide) does not affect the while() loop.

#846 [Explained](#) [Report](#) [Bookmark](#)

An Identifier can start with.?

- A Alphabet
- B Underscore ( \_ ) sign
- C Any character that can be typed on a keyboard
- D Option A & Option B

Correct Answer :D

# Explanation

Identifier is just a name given to a Function, Variable etc. Identifier name should contain only Letter, Numbers and Underscore.

#847 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program.?

```
int main()
{
 struct laptop
```

```

 int cost;

 char brand[10];

 };

struct laptop L1={5000,"ACER"};

struct laptop L2={6000,"IBM"};

printf("Name=%s",L1.brand);

return 0;
}

```

- **A**CER
- **B**IBM
- **C**Compiler error
- **D**None of the above

**Correct Answer :A**

## Explanation

**You can initialize structure members at the time of creation of Structure variables.**

**#848**[Explained](#) [Report](#) [Bookmark](#)

**What is the number of characters used to distinguish Identifier or Names of Functions and Global variables.?**

- **A**31

- **B32**
- **C33**
- **D28**

Correct Answer :A

## Explanation

First 31 characters in general. If first 31 characters are same for two different identifiers, compiler gets confused.

#849 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with structures pointers.?

```
int main()
{
 struct forest
 {
 int trees;
 int animals;
 }F1,*F2;
 F1.trees=1000;
 F1.animals=20;
```

```
F2=&F1;

printf("%d ",F2.animals);

return 0;

}
```

- A 0
- B 20
- C Compiler error
- D None of the above

Correct Answer :C

## Explanation

F2.animal is not allowed as F2 is a pointer to structure variable. So use ARROW operators. F2->animal.

#850 [Explained](#) [Report](#) [Bookmark](#)

Find a Floating Point constant.

- A 12.3E5
- B 12e34
- C 125.34857
- D All the above.

Correct Answer :D

## Explanation

**Floating Point can be represented in two forms.** 1. **Fractional Form eg. 12345.67** 2. **Exponential Form (Mantissa)e(number) or (Mantissa)E(number) eg. 123.4567E2 (e2 = 10 power 2 = 100)**

#851 [Explained](#) [Report](#) [Bookmark](#)

**What is the output of C program with structures.?**

```
int main()

{
 struct tree

 {
 int h;

 int w;

 };

 struct tree tree1={10};

 printf("%d ",tree1.w);

 printf("%d",tree1.h);

 return 0;
}
```

- A 0 0
- B 10 0

- C 0 10
- D 10 10

Correct Answer :C

## Explanation

struct tree tree1={10}; Assigns the value to corresponding member. Remaining are set to Zero. So w is zero.

#852 [Explained](#) [Report](#) [Bookmark](#)

Which of the following cannot be checked in a switch-case statement?

- A Character
- B Integer
- C Float
- D enum

Correct Answer :C

## Explanation

The switch/case statement in the c language is defined by the language specification to use an int value, so you can not use a float value. switch( expression ) { case constant-expression1: statements 1; case constant-expression2: statements 2; case constant-expression3: statements3 ; ... ... default : statements 4; } The value of the 'expression' in a switch-case statement must be an integer, char, short, long. Float and double are not allowed.

## What is the output of C program.?

```
int main()
{
 struct book
 {
 int pages;
 char name[10];
 }a;
 a.pages=10;
 strcpy(a.name,"Cbasics");
 printf("%s=%d", a.name,a.pages);
 return 0;
}
```

- A Empty string=10
- B C=basics
- C Cbasics=10
- D Compiler error

Correct Answer :B

# Explanation

pages and name are structure members. a is a structure variable. To refer structure members use a DOT operator say a.name.

#854 [Explained](#) [Report](#) [Bookmark](#)

What is the output of C program with structures.?

```
int main()
{
 struct tree
 {
 int h;
 int rate;
 };
 struct tree tree1={0};

 printf("%d ",tree1.rate);

 printf("%d",tree1.h);

 return 0;
}
```

- A 0 0
- B -1 -1
- C NULL NULL
- D Compiler error

Correct Answer :A

## Explanation

Easiest way of initializing all structure elements. `struct tree tree1={0};`

#855 [Explained](#) [Report](#) [Bookmark](#)

What is length of an Identifier that is unique for Non Global Variables and Non Function Names.?

- A 32
- B 63
- C 64
- D 68

Correct Answer :B

## Explanation

if 31 is present choose. Because old compilers support up to 31 only. Upto first 63 characters you can show differentiation in the name of say `int abcdefghijklmnopqrstuvwxyz1234567788 = 10;` `int abcdefghijklmnopqrstuvwxyz1234567799 = 20;`

#856 [Explained](#) [Report](#) [Bookmark](#)

## What will be the output of the program?

```
#include<stdio.h>

int main()

{
 int i=4, j=-1, k=0, w, x, y, z;

 w = i || j || k;

 x = i && j && k;

 y = i || j && k;

 z = i && j || k;

 printf("%d, %d, %d, %d\n", w, x, y, z);

 return 0;
}
```

- A1, 1, 1, 1
- B1, 1, 0, 1
- C1, 0, 0, 1
- D1, 0, 1, 1

Correct Answer :D

# Explanation

Step 1: int i=4, j=-1, k=0, w, x, y, z; here variable i, j, k, w, x, y, z are declared as an integer type and the variable i, j, k are initialized to 4, -1, 0 respectively. Step 2: w = i || j || k; becomes w = 4 || -1 || 0;. Hence it returns TRUE. So, w=1 Step 3: x = i && j && k; becomes x = 4 && -1 && 0; Hence it returns FALSE. So, x=0 Step 4: y = i || j && k; becomes y = 4 || -1 && 0; Hence it returns TRUE. So, y=1 Step 5: z = i && j || k; becomes z = 4 && -1 || 0; Hence it returns TRUE. So, z=1. Step 6: printf("%d, %d, %d, %d\n", w, x, y, z); Hence the output is "1, 0, 1, 1".

#857 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about the program?

```
#include<stdio.h>

int main()

{
 int x = 30, y = 40;

 if(x == y)

 printf("x is equal to y\n");

 else if(x > y)

 printf("x is greater than y\n");
```

```
else if(x < y)

 printf("x is less than y\n")

return 0;

}
```

- A**Error: Statement missing**
- B**Error: Expression syntax**
- C**Error: Lvalue required**
- D**Error: Rvalue required**

Correct Answer :A

## Explanation

This program will result in error "Statement missing ;" printf("x is less than y\n") here ; should be added to the end of this statement

#858**Explained Report Bookmark**

What is the size of a C structure.?

- A  
**C structure is always 128 bytes.**
- B  
**Size of C structure is the total bytes of all elements of structure.**
- C  
**Size of C structure is the size of largest element.**

- D
- None of the above**

Correct Answer :B

## Explanation

**Individually calculate the sizes of each member of a structure and make a total to get Full size of a structure.**

#859 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
int main()
{
 char *str;
 str = "%d\n";
 str++;
 str++;
 printf(str-2, 300);
 return 0;
}
```

- A  
No output
- B  
30
- C  
3
- D  
300

Correct Answer :D

## Explanation

Twice str is incremented n then when we do str-2, it is again pointing to starting position. So str contains "%d\n".

So printf statement would be like printf("%d\n",300);

Hence output will be 300.

#860 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

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

```
int main()
```

```

{
 int a[3][4] = {1, 2, 3, 4, 4, 3, 2, 1, 7, 8, 9, 0};

 printf("%u, %u\n", a+1, &a+1);

 return 0;
}

```

- A65474, 65476
- B65480, 65496
- C65480, 65488
- D65474, 65488

Correct Answer :B

## Explanation

Step 1: `int a[3][4] = {1, 2, 3, 4, 4, 3, 2, 1, 7, 8, 9, 0};` The array `a[3][4]` is declared as an integer array having the 3 rows and 4 columns dimensions. Step 2: `printf("%u, %u\n", a+1, &a+1);` The base address(also the address of the first element) of array is 65472. For a two-dimensional array like a reference to array has type "pointer to array of 4 ints". Therefore, `a+1` is pointing to the memory location of first element of the second row in array a. Hence  $65472 + (4 \text{ ints} * 2 \text{ bytes}) = 65480$  Then, `&a` has type "pointer to array of 3 arrays of 4 ints", totally 12 ints. Therefore, `&a+1` denotes " $12 \text{ ints} * 2 \text{ bytes} * 1 = 24 \text{ bytes}$ ". Hence, begining address  $65472 + 24 = 65496$ . So, `&a+1 = 65496` Hence the output of the program is 65480, 65496

#861 Explained Report Bookmark

The library function used to reverse a string is

- **A**strrstr()
- **B**strrev()
- **C**revstr()
- **D**strreverse()

Correct Answer :B

## Explanation

**strrev(s)** Reverses all characters in s Example: #include **#include int main(void) { char \*str = "IndiaBIX"; printf("Before strrev(): %s\n", str); strrev(str); printf("After strrev(): %s\n", str); return 0; }** Output: Before strrev(): IndiaBIX After strrev(): XI~~B~~aidnl

#862 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

```
#include<stdio.h>

#include<string.h>

int main()

{

 char sentence[80];

 int i;
```

```
printf("Enter a line of text\n");

gets(sentence);

for(i=strlen(sentence)-1; i >=0; i--)

 putchar(sentence[i]);

return 0;

}
```

- **A**  
The sentence will get printed in same order as it entered
- **B**  
The sentence will get printed in reverse order
- **C**  
Half of the sentence will get printed
- **D**  
None of above

Correct Answer :B

## Explanation

The C library function `char *gets(char *str)` reads a line from `stdin` and stores it into the string pointed to by `str`. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.

**The `strlen()` function takes a string as an argument and returns its length. The returned value is of type `size_t` (the unsigned integer type).**

**The for loop iterated from length to 0. Therefore string/ sentence will be printed in reverse order**

#863 [Explained](#) [Report](#) [Bookmark](#)

**What will be the output of the program ?**

```
#include<stdio.h>

struct course

{
 int courseno;

 char coursename[25];

};

int main()

{
 struct course c[] = { {102, "Java"},

 {103, "PHP"} ,

```

```

 {104, "DotNet"} } ;

printf("%d ", c[1].courseno);

printf("%s\n", *(c+2)).coursename);

return 0;

}

```

- A  
103 DotNet
- B  
102 Java
- C  
104 DotNet
- D  
103 PHP

**Correct Answer :A**

## Explanation

The `c[1].courseno` points to 103

In `(*(c+2)).coursename` , `*(c+2)` is same as writing :

`c[2]` that is `c[2]` or `*(c+2)` points to `{104, "DotNet"}`

Therefore `(*(c+2)).coursename` or u can say `c[2].coursename` will give `DotNet`.

So final Answer will be 103 DotNet

#864 [Explained](#) [Report](#) [Bookmark](#)

Point out the error in the program?

```
struct emp

{

 int ecode;

 struct emp *e;

};

• AError: in structure declaration
• BLinker Error
• CNo Error
• DNone of above
```

Correct Answer :C

## Explanation

This type of declaration is called as self-referential structure. Here \*e is pointer to a struct emp.

#865 [Explained](#) [Report](#) [Bookmark](#)

What do the following declaration signify?

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

- A f is a pointer variable of function type.
- B f is a function returning pointer to an int.
- C f is a function pointer.
- D f is a simple declaration of pointer variable.

Correct Answer :B

## Explanation

f is a function returning pointer to an *int*.

#866 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

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

```
int main() {

 static char *s[3]={"math","phy","che"};

 typedef char *(*ppp)[3];

 static ppp p1=&s,p2=&s,p3=&s;

 char * (*array[3]))[3]={&p1,&p2,&p3};
```

```

char * (*(*(*ptr) [3])) [3]=&array;

p2+=1;

p3+=2;

printf("%s", (**ptr[0]) [2]);

return 0;

}

```

- A  
math
- B  
che
- C  
phy
- D  
Compiler error

**Correct Answer :B**

## Explanation

Here ptr is pointer to array of pointer to string. P1, p2, p3 are pointers to array of string. array[3] is array which contain pointer to array of string.

**Note:** In the above figure upper part of box represent content and lower part represent memory address. We have assumed arbitrary address.

As we know  $p[i]=*(p+i)$

`(***ptr[0])[2]=(*(**ptr+0))[2]=(***ptr)[2]`

`=(**(&array))[2] //ptr=&array`

`=(**array)[2] //From rule *&p=p`

`=(**(&p1))[2] //array=&p1`

`=(*p1)[2]`

`=(*&s)[2] //p1=&s`

`=s[2]=""che"`

**#867** [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program, if a short int is 2 bytes wide?

```
#include <stdio.h>

int main()

{
 short int i = 0;

 for(i<=5 && i>=-1; ++i; i>0)

 printf("%u, ", i);
```

```
 return 0;
}
```

- A  
1 ... 65535
- B  
Expression syntax error
- C  
No output
- D  
0, 1, 2, 3, 4, 5

Correct Answer :A

## Explanation

**for( $i \leq 5 \ \&\& \ i \geq -1$ ;  $++i$ ;  $i > 0$ )** so expression  $i \leq 5 \ \&\& \ i \geq -1$  initializes for loop. expression  $++i$  is the loop condition.

**expression  $i > 0$  is the increment expression.**

In **for(  $i \leq 5 \ \&\& \ i \geq -1$ ;  $++i$ ;  $i > 0$ )** expression  $i \leq 5 \ \&\& \ i \geq -1$  evaluates to one.

**Loop condition always get evaluated to true.**

Also at this point it increases  $i$  by one. An increment\_expression  $i > 0$  has no effect on value of  $i$ .so for loop get executed till the limit of integer (ie. 65535)

#868 [Explained](#) [Report](#) [Bookmark](#)

**C is very fast to execute and safe to embed along with microprocessors. Device drivers are written in C and C++.**

- APrimary Constants
- BSecondary Constants
- CBasic Constants and Advanced Constants
- DPrimary Constants and Secondary Constants

Correct Answer :D

## Explanation

**Primary Constants are Integer (int), Floating Point (float) , Character (char) Secondary Constants are Structure, Union, Array and Enum.**

#869 [Explained](#) [Report](#) [Bookmark](#)

Find an integer constant.

- A3.145
- B34
- C"125"
- DNone of the above

Correct Answer :B

## Explanation

**Integer constant is a full or whole number without any decimal point. So 3.14 is a floating point number or Real number.**

#870 [Explained](#) [Report](#) [Bookmark](#)

## What is the output of C program with structures.?

```
int main()

{

 struct ship

 {

 char color[10];

 }boat1, boat2;

 strcpy(boat1.color,"RED");

 printf("%s ",boat1.color);

 boat2 = boat1;

 strcpy(boat2.color,"YELLOW");

 printf("%s",boat1.color);

 return 0;

}
```

- A RED RED
- B RED YELLOW
- C YELLOW YELLOW
- D Compiler error

Correct Answer :A

# Explanation

**boat2=boat1 copies only values to boat2 memory locations. So changing boat2 color does not change boat1 color.**

#871 Explained Report Bookmark

**What is the output of C program with structures.?**

```
int main()
{
 struct tree
 {
 int h;
 }

 struct tree tree1;

 tree1.h=10;

 printf("Height=%d",tree1.h);

 return 0;
}
```

- A Height=0
- B Height=10

- CHeight=
- DCompiler error

Correct Answer :B

## Explanation

Notice a missing semicolon at the end of structure definition. struct tree { int h; };

#872 [Explained](#) [Report](#) [Bookmark](#)

A C Structure or User defined data type is also called.?

- ADerived data type
- BSecondary data type
- CAggregate data type
- DAll the above

Correct Answer :D

## Explanation

NaN

#873 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program ?

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

```
#include<string.h>
```

```
int main()

{

 char sentence[80];

 int i;

 printf("Enter a line of text\n");

 gets(sentence);

 for(i=strlen(sentence)-1; i >=0; i--)

 putchar(sentence[i]);

 return 0;

}
```

- A The sentence will get printed in same order as it entered
- B The sentence will get printed in reverse order
- C Half of the sentence will get printed
- D None of above

Correct Answer :B

# Explanation

```
#include<stdio.h>

#include<string.h>

int main()

{

 char sentence[80];

 int i;

 printf("Enter a line of text\n");

 gets(sentence); //gets function is used to scan or read a
line of text from a standard input (stdin)

 //The strlen() function calculates the length of a given
string

 for(i=strlen(sentence)-1; i >=0; i--) // here we are
iterating in reverse order

 putchar(sentence[i]); //putchar writes a single
character and print reverse order characters by index

 return 0;

}
```

#874 [Explained](#) [Report](#) [Bookmark](#)

Program which is responsible to process code before compiler is

null

- A Assembler
- B Debugger
- C Preprocessor
- D Linker

Correct Answer :C

## Explanation

Four Steps of Compilation: preprocessing, compiling, assembly, linking.

*Preprocessing:*

Preprocessing is the first step. The preprocessor obeys commands that begin with # (known as directives) by:

- removing comments
- expanding macros
- expanding included files

If you included a header file such as #include <stdio.h>, it will look for the stdio.h file and copy the header file into the source code file.

The preprocessor also generates macro code and replaces symbolic constants defined using #define with their values.

### ***Compiling:***

**Compiling** is the second step. It takes the output of the preprocessor and generates assembly language, an intermediate human readable language, specific to the target processor.

### ***Assembly:***

**Assembly** is the third step of compilation. The assembler will convert the assembly code into pure binary code or machine code (zeros and ones). This code is also known as object code.

### ***Linking:***

**Linking** is the final step of compilation. The linker merges all the object code from multiple modules into a single one. If we are using a function from libraries, linker will link our code with that library function code.

In static linking, the linker makes a copy of all used library functions to the executable file. In dynamic linking, the code is not copied, it is done by just placing the name of the library in the binary file.

#875 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```

#include<stdio.h>

int main()

{

 signed int x = -10;

 unsigned int y = 10,z;

 y = y >> 2;

 z = y > x ? !y : ~x ;

 printf("%u",z);

}

```

- A  
4294967285
- B  
10
- C  
-10
- D  
9

**Correct Answer :D**

## Explanation

**y = y >> 2 assign value 2 to y**

**z = y > x -> here we checked y > z so it will false and execute the ~x**

**therefore z =~x (value of ~x is 9 )**

**output is 9**

**#876Explained Report Bookmark**

**Find the output of the following:**

```
#include<stdio.h>

int main()
{
 char s1[] = "hear will let you go ahead";
 strcat(s1, "\b\b\b\b\behind\rf");
 printf(s1);
}
```

- A  
**fear will let you go behind**
- B  
**behind will let you go ahead**
- C  
**hear will let you go ahead**
- D  
**CompileTime Error**

**Correct Answer :A**

## **Explanation**

\b will move the cursor back. Since here we are using 5 \b then the cursor will move from 'd' of 'ahead' to 'a' and merge the previous string with 'behind'. \r will point to the first cursor i.e. 'h' of 'hear' and replace it with 'f'.

#877 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
#include<stdio.h>

int main()

{
 char c1='1' , c2 = '2',c3;

 c3 = 48 != c1 < c2 && c2++;

 printf("%c",c3+48);

}
```

- A  
50
- B  
0
- C  
1
- D  
51

Correct Answer :C

## Explanation

**ASCII character Value of c1 = 49 and c2 = 50**

**when below condition checked**

**c3 = 48 != c1 < c2 && c2++; now**

**c3 = 48 != 49 < 50 && 51 (c2 incremented by 1 )**

**so final result in c3 is 1 as all condition are true then**

**printf("%c",c3+48); here 1 + 48 = 49**

**here are printing %c character value of 49 is 1**

**#878Explained Report Bookmark**

**Find the output of the following:**

```
#include<stdio.h>

int main()

{
 float f = 1.30;

 int i=5;

 while(i--)
 {
```

```

 f = f - 0.01;

 }

 if(1.25 == f)

 printf("True ");

 else

 printf("False");

}

```

- **A**  
Infinite Loop
- **B**  
No Output
- **C**  
False
- **D**  
True

Correct Answer :D

## Explanation

```

#include<stdio.h>

int main()

{

 float f = 1.30;

```

```

int i=5;

while(i--) // Execute Till i value become 0

{
 f = f - 0.01; // Everytime we are
 subtracting float by by 0.01

}

// the final value of f after loop is 1.250000
and it satisfy the if condition so it will print TRUE

if(1.25 == f)

 printf("\n True ");

else

 printf("False");

}

```

**#879** [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```

#include<stdio.h>

void fun_enum(DEPARTMENT d)

{

```

```
 printf("%d", d);

}

int main()

{

 typedef enum {TCT, ADMN, MRKT, SAN}DEPARTMENT;

 fun_enum(SAN);

}

• A
 0
• B
 1
• C
 Garbage
• D
 Compile Time Error
```

Correct Answer :D

## Explanation

Error : unknown type name 'DEPARTMENT'

because enum is declared inside the main function scope is limited

**Correct Program is**

```
#include<stdio.h>

typedef enum d {TCT,ADMN,MRKT,SAN}DEPARTMENT;

void fun_enum(DEPARTMENT d)

{

 printf("%d",d);

}

int main()

{

 fun_enum(SAN);

}
```

**#880**Explained Report Bookmark

**Find the output of the following:**

```
void fun_test(int a,int b)
```

```

{

 {

 int a = 5;

 ++a,b++,a++;

 }

}

int main()

{

 int a=2,b=3;

 fun_test(a,b);

 printf("%d %d",a,b);

}

- A
2 3
- B
7 4
- C
3 4
- D
Garbage 4

```

**Correct Answer :A**

# Explanation

Here parameters are passed to a function fun\_test by value. So whatever modification ( increment operation ) is performed on data in fun\_test will not change the variable values in main().

#881 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following

```
#include<stdio.h>

void test_rec(int *k)

{
 if(k==NULL)
 return ;

 *k = *k + 10 ;

 test_rec(--k) ;

}

int main()

{
 register int a[] = {5,6,7,8,9} ;
```

```
 test_rec(((int *)(&a+1))-1);

 printf("%d",a[0]);

}
```

- A  
Error
- B  
5
- C  
Garbage
- D  
15

Correct Answer :A

## Explanation

Here the address of the variable is requested which can't be done because we can't use the address of a variable declared register. It tells the compiler to try to use a CPU register, instead of RAM, to store the variable

#882 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following

```
int fun_array(const char *p)

{

 ++p;
```

```

 ++*p;

}

int main()

{
 char arr[] = {'A','B','C','D','E','F'};

 fun_array(arr);

 printf("%c %c %c",arr[0],arr[1],arr[2]);
}

```

- A  
C B D
- B  
C C C
- C  
A C C
- D  
Compile Time Error

Correct Answer :D

## Explanation

\*p is a read only location i.e direct pointer access is asked here in order to increment it. And as per rule any read only location cant be updated.

#883 [Explained](#) [Report](#) [Bookmark](#)

**Find the output of the following**

```
#include<stdio.h>

int main()
{
 char s1[] = "PreCAT";
 char s2[] = "Sunbeam";
 char *p1=s1,*p2=s2;
 while(*p1)
 {
 p1++;
 p2++;
 }
 printf("%s",p2);
}
```

- A  
Sunbeam
- B  
No Output

- C  
m
- D  
Tm

Correct Answer :C

## Explanation

\*p1 =s1 means pointer p1 is pointing to base address of s1 ( i.e pointing to letter P)

Each time in while loop \*p1 and \*p2 incremented. The while loop condition remains true for 6 times because the “ PreCat” has 6 letters and the pointer \*p1 and \*p2 also incremented for 6 times. Now at 7th iteration \*p1 is pointing to NULL and while loop condition falses. But the \*p2 will point to the last letter in array s2 i.e m. Therefore ans is m

While loop            1      2      3      4      5      6

\*p1                    r      e      C      a      t      NULL

(iteration Fail)

\*p2                  u      n      b      e      a      m

#884 Not Explained Report Bookmark

Find the output of the following

```
typedef union

{
 unsigned b1 : 4;
 signed b2 : 2;

} TEST;

int main()

{
 TEST t1={0};

 t1.b1 >> 2;

 printf("%d",t1.b1);

}
```

- A  
1
- B  
0

- C  
-1
- D  
2

Correct Answer :B

## No Explanation Available

#885 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following

```
#include<stdlib.h>

int main()

{
 char
*arr[]={ "Sunbeam" , "Karad" , "Pune" , "Hinjawadi" } ;

 char **p1 = arr+3;

 char *p2 =++*p1;

 printf("%s",++p2);

}
```

- A  
eam

- **B**  
njawadi
- **C**  
Ceam
- **D**  
Ojawadi

Correct Answer :B

## Explanation

`**p1 = arr +3` ---> `**p1` is pointing to base value of Hinjewadi i.e. H

`*p2 = ++ *p1` ----> incrementing `*p1` --- > Now it will point to letter i

-----> `*p2` is pointing to character i

`++p2` -----> now p2 is incremented. It will point to letter n

In `printf()` the `%s` format specifier is used so it will print the string starting from letter n till NULL characters are encountered.

#886 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following if values are entered as ---- BillGates 56 78

```
struct student
```

```
{
```

```
 char *s;
```

```

 int marks[2];

};

int main()

{

 struct student s1;

 scanf("%s %d
%d", s1.s, &s1.marks[0], &s1.marks[1]);

 printf("%s %d
%d", s1.s, s1.marks[0], s1.marks[1]);

}

```

- A BillGates 56 78
- B CompileTime Error
- C Runtime Error
- D None of these

**Correct Answer :A**

## Explanation

**s is a character pointer. Marks is array of 2 integer**

**S1 is an object created for structure student.**

**Through s1 we can access the structure variables \*s and marks[0] and marks[1].**

**s1.s will hold a character string and s1.marks[0] and s1.marks[1] will hold 2 integer values.**

**Whatever values entered through scanf() will be printed on screen.  
There is no any compiler or runtime error in this code.**

**#887** [Explained](#) [Report](#) [Bookmark](#)

**Find the output of the following**

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int main()

{

 char *c1,*c2 ;

 c1 =(char *) malloc(sizeof(char)*10) ;

 strcpy(c1,"Hinjawadi") ;

 c2 ="Pune" ;
```

```
 free(c2);

 c1 = NULL;

 c2 = NULL;
}
```

- A      **Memory Leak**
- B      **Dangling Pointer**
- C      **CompileTime Error**
- D      **None of these**

Correct Answer :D

## Explanation

Here some might think that dangling pointer is the answer, but it would have been the case if `c1=NULL` and `c2=NULL` statements were not used in the program. Since we are assigning the `NUL` value to the '`c1`' and '`c2`', they will not point to the deleted memory.

**Memory leak is not an option either because in order to avoid memory leak pointer must always be freed using `free()`.**

#888 [Explained](#) [Report](#) [Bookmark](#)

Find the correct statement from the following:

- A      **fread function can read only binary data from the file**

- **B**  
fread can not be used to read 1 byte at a time.
- **C**  
fread can not read multiple records in one array in single call
- **D**  
fread function can read text / binary data from file.

Correct Answer :D

## Explanation

The fread() function is the complementary of fwrite() function. fread() function is commonly used to read binary data. It accepts the same arguments as fwrite() function does. The syntax of fread() function is as follows:

**Syntax:** size\_t fread(void \*ptr, size\_t size, size\_t n, FILE \*fp);

The ptr is the starting address of the memory block where data will be stored after reading from the file. The function reads n items from the file where each item occupies the number of bytes specified in the second argument. On success, it reads n items from the file and returns n. On error or end of the file, it returns a number less than n.

#889 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the following C code?

```
int main()
{
 FILE *fp = fopen("abc.txt", "w+");
}
```

```

int a,b,c;

fprintf(fp,"%d %d %d",10,20,30);

rewind(fp);

printf("%d",fscanf(fp,"%d %d %d",&a,&b,&c));

fclose(fp);

}

```

- A  
10 20 30
- B  
3
- C  
1
- D  
-1

**Correct Answer :B**

## Explanation

. abc.txt file will be created if exist or created if it doesn't exist with write permission.

Decimals 10,20,30 will be printed in file pointed by fp pointer using **fprintf()** .

rewind function sets the file position to the beginning of the file for the stream pointed to by stream.

**fscanf()** will scan the data from file pointed by fp pointer and print on screen.

[Note : The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file]

**fclose()** will close the file pointed by fp pointer.

#890 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
typedef struct employee

{
 int id;
 char dept[10];
 struct employee *emplist;

}EMP;

int main()

{
 EMP e1={101,"ADMN",NULL};
 EMP e2={102,"MRKT",NULL};
 EMP e3={103,"TCT",NULL};
```

```

 EMP e4={104,"SAN",NULL};

 e1.emplist = &e2 ;

 e2.emplist = &e3;

 e3.emplist = &e4;

 printf("%d",e1.emplist->emplist->emplist->id);

}

```

- A  
101
- B  
102
- C  
NULL
- D  
104

Correct Answer :D

## Explanation

**structure employee will allocate space to int id, char dept[10], struct employee \*emplist with 'EMP' object.**

**EMP e1={101,"ADMN",NULL}; and so will add the values to variables of 'EMP' object of the structure employee.**

**e1.emplist = &e2 ; and so will create a connection with the next element. Here next element to e1 will be e2.**

e1.emplist->emplist->emplist->id will print id of element in such a way that is pointed 4th in the list i.e. 104 in our case.

#891 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

Find the output of the following:

```
int main()

{

 float farr[] = {1.1,2.2,3.3,4.4,5.5};

 float *fp = farr+4;

 int x ;

 printf("%.2f",fp[-(x=sizeof(5.5)-6)]);

}
```

- A Garbage
- B 4.40
- C 3.30
- D Compile Time Error

Correct Answer :C

## Explanation

```
float *fp = farr+4;
```

farr+4 i.e. farr[4] will be pointed by fp pointer of float type

sizeof(5.5) is 8 since it is of a float type. Thus x=sizeof(5.5)-6 will result in 8-6 i.e. 2 which is held by x variable..

in fp[-x] i.e. in fp[-2] -2 means access the 2nd last value held by farr array which is 3.3. But since we are printing %.2f which indicates printing 2 decimal values after '.'. Thus result will be 3.3 and added '0' which makes it 3.30

#892 Not Explained Report Bookmark

Find the output of the following:

```
void fun_test(char *c)
```

```
{
```

```
++*++c;
```

```
}
```

```
int main()
```

```
{
```

```
int num = 678;
```

```
fun_test(&num);
```

```
printf("%d",num);
```

```
}
```

- A  
678
- B  
934
- C  
Garbage
- D  
0

Correct Answer :B

## No Explanation Available

#893 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
void fun(int *const p)
```

```
{
```

```
 ++*p;
```

```
}
```

```
int main()
```

```
{
```

```
 auto int a_1 =10;
```

```
{
```

```
 auto int a_1 =2;
```

```
 fun(&a_1);

}

printf("%d",a_1);
}
```

- A  
10
- B  
11
- C  
Garbage
- D  
Compile Time Error

Correct Answer :A

## Explanation

it is clear that a\_1 will print 10 since the calling of fun function is done locally i.e. all the operation done is within auto int a\_1 =10;

```
{

auto int a_1 =2;

fun(&a_1);

}
```

#894 [Explained](#) [Report](#) [Bookmark](#)

**Find the output of the following:**

```
int main()

{

 char ch='z';

 static int n=3;

 if(n)

 {

 ch >> 1 != ch >> 1 ^ n--;

 main();

 }

 printf("%d ",ch<<1);

}
```

- **A**  
Runtime Error
- **B**  
122 122 122 122
- **C**  
244 244 244 244
- **D**  
0 0 0 0

**Correct Answer :C**

# Explanation

ASCII value for 'z' is 122. If block is basically used for conditioning. Here n will work as a counter from 3 to 0. Each time main will be called and `printf("%d ",ch<<1);` will be executed depending on the counter n. `ch<<1` is a bitwise shift operator when each bit is shifted to the left by 1 position which will result in `ch=244`.

#895 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
int fun_test(int a,int b)

{
 a= !a;

 b = ~b;

 return a,b;
}

int main()

{
 int a=4,b=5;

 a,b = fun_test(!a,~b);

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

}

- A  
0 -5
- B  
4 5
- C  
5 5
- D  
4 4

Correct Answer :B

## Explanation

In main !a results in 0 and ~b results in -6 which are passed to fun\_test() function.

Inside function a will hold 0 and b will hold -6 thus performing ! and ~ on the respectively will result in 1 and 5.

Return a,b will return the value of variables b only thus resulting a will hold the value from main and b will hold the value returned from function i.w. a=4 and b=5

#896 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
int main()

{

 typedef enum {O1=2 >= 5 != 4 ,O2= !(2 < 5 != 4) } OPR;
```

```

OPR choice=0;

do

{

 switch(choice)

 {

 case 01:

 --choice;

 printf("%2d",choice); break;

 case 02:

 printf("%2d",choice); break;

 }

 break;

} while (++choice);

}

```

- **A**  
Compile Time Error
- **B**  
1 0
- **C**  
0

- D  
1

Correct Answer :C

## Explanation

```
typedef enum {O1=2 >= 5 != 4 ,O2= !(2 < 5 != 4)} OPR;
```

Here we are creating a enum list where condition will be checked and stores the result in variable such as in this case ( $2 < 5 \neq 4$ ) would result in 1 because the condition results in true. Same for  $2 \geq 5 \neq 4$  condition which results in false and will be stored in O1.

OPR will indirectly hold the value for num elements.

OPR choice=0; will declare the choice with 0 value but during execution expected true value will be transferred to choice i.e. from O2.

When we enter do while loop where  $++choice$  is the condition increment, notice that there is a break mentioned between switch and while statement which causes the system to come out of the current loop.

As O2 holds the true value switch statement executed the block of code mentioned under case O2 section.

As choice is holding true which is 0 in boolean form, it will be printed on screen

Find the output of the following:

```
int main()

{

 unsigned char ch=255;

 unsigned int val = ~!(ch >> 7) ? ch << 1 : !ch ;

 printf("%u",val);

}
```

- A  
-1
- B  
510
- C  
0
- D  
127

Correct Answer :B

## Explanation

ch>>7 results in 1 after performing shift operation of binary form of 255 which is 11100001. Thus !(ch>>7) would be 0 where ! checks whether value or condition results in true or false. After performing ! operation the result comes out to be 4294967295. Which is true in boolean form. Thus ch<<1 gets executed resulting in 510

#898 Explained Report Bookmark

**Find the output of the following:**

```
int main()
{
 int a=2,b=3,c=4;

 (c>>=1,b<<1) ? c/=2 : b<=2;

 printf("%d",c);
}
```

- A  
2
- B  
6
- C  
1
- D  
0

**Correct Answer :C**

## **Explanation**

**c>>1**

**C which is 4 in decimal and 0100 in binary will be shifted by 1 bit to it's right which will resulting in 0010 in binary in 2 in decimal**

**b<<1**

**B** which is b in decimal and 0011 in binary will be sifted by 1 bit to it's right resulting in 0110 in binary in 6 in decimal

c=2

**C** which is now 2 will be divided by 2 and result 1 will be stored in c only.

We then print value of c which is 1.

#899 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
#include <stdio.h>

int main()
{
 int d;

 d = printf("ccatpreparation") ==
printf("\rccatpreparation") ;

 printf("%2d",d);
}
```

- **A**  
ccatpreparation 1
- **B**  
ccatpreparation 2
- **C**  
ccatpreparation 0

- D  
ccatpreparation ccatpreparation 0

Correct Answer :C

## Explanation

d = printf("ccatpreparation ") == printf("\rccatpreparation ") ;

As we are using \r , here the test written in the right most printf statement i.e. “printf("\rccatpreparation ")” will be printed. Then it will be compared with next printf statement i.e. “printf("ccatpreparation ")” since both print statements contains different values false i.e. 0 will be return to d.

#900 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following:

```
int main()
{
 int a=5,b=6;
 char c='A';
 float f=2.3;
 printf("%d",sizeof(a+f) / sizeof('A') << 2);
}
```

- A 8
- B 4
- C 2
- D 1

Correct Answer :B

## Explanation

`sizeof(a+f)`

**Size of a i.e. 2Byte and size of f i.e. 2Byte will be added, resulting 4Byte**

`sizeof('A')`

**Size of char i.e. 4Byte**

`sizeof('A') << 2`

**Size of char is 4 i.e. in binary it will be 0100. When we perform <<2 we shift bits to left size and keep adding 0's from right end. Thus result will be 0001 0000 i.e. 16**

**We have this expression - `sizeof(a+f) / sizeof('A') << 2`**

**here the tricky part is first divide will happen ( check precedence table ) so**

`sizeof(a+f) = 4 / sizeof('A') = 4` so result is 1

then we have `1 << 2` that result to 4

#901 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program given below?

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

```
main()
```

```
{
```

```
 char *x = NULL;
```

```
 printf("%c", *x);
```

```
}
```

- A  
0
- B  
NULL
- C  
Compile error
- D  
Runtime error

Correct Answer :D

## Explanation

**In the program x points the NULL address. It is invalid to access the NULL address hence the program gives Runtime error.**

**Therefore the output of the program is Runtime error.**

**#902** [Explained](#) [Report](#) [Bookmark](#)

**Which statement is correct about the program given below?**

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

```
int main()
```

```
{
```

```
 int j=10;
```

```
 int *i=&j;
```

```
 return 0;
```

```
}
```

- **A**  
j is a pointer to an int and stores address of i
- **B**  
i is a pointer to a pointer to an int and stores address of j
- **C**  
i and j are pointers to an int
- **D**  
i is a pointer to an int and stores address of j

**Correct Answer :D**

# Explanation

In program 'i' is the variable contain pointer. So it is pointer variable and points toward an integer type in memory location. Therefore 'i' is a pointer to an int.

Now the address of 'j' is assigned to the 'i' pointer, i.e. address of 'j' store to 'i' location.

Therefore the 'i' is a pointer to an int and it stores the address of 'j'.

#903 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the program given below?

```
#include<stdio.h>

main()
{
 char *p= "Xyz";
 while(*p)
 printf("%c", *p++);
}
```

- A  
Xyz
- B  
yz

- **C**  
Runtime error
- **D**  
Compile error

Correct Answer :A

## Explanation

In program while loop continues until \*s is not equal to '\0'. Inside loop character is fetched first and address is incremented later.

Therefore the print statement i.e. printf("%c", \*p++); will print the Xyz in output.

#904 [Explained](#) [Report](#) [Bookmark](#)

Which format specifier is used for printing double value?

null

- **A**  
%Lf
- **B**  
%L
- **C**  
%lf
- **D**  
None of the above

Correct Answer :C

## Explanation

**The %lf format specifier is used for printing the double value in a C program.**

#905 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statement is used to free the allocated memory space for a program

null

- A  
vanish(var-name);
- B  
remove(var-name);
- C  
erase(var-name);
- D  
free(var-name);

Correct Answer :D

## Explanation

**The memory allocated by malloc(), calloc(), or realloc() function is deallocated by using the library function free(var-name).**

#906 [Explained](#) [Report](#) [Bookmark](#)

Which of the following are declarations?

1. float square (float x){?}
  2. double pow(double, double);
  3. extern int x;
- A  
1

- **B**  
1 and 3
- **C**  
2
- **D**  
1 and 2

Correct Answer :B

## Explanation

**double pow(double, double);** instruction is a function prototype declaration

**extern int x;** instruction is an external variable declaration.

Therefore 1 and 3 are declarations and 2 is definition.

#907 [Explained](#) [Report](#) [Bookmark](#)

What is the correct value returned to the operating system upon successful completion of a program?

- **A**  
-1
- **B**  
1
- **C**  
Programs do not return a value
- **D**  
After successful completion of a program, 0 is returned to the operating system.

Correct Answer :D

## Explanation

After successful completion of a program 0 is returned to the operating system.

Your `main()` function should return 0 on success.

If you call `exit()`, you can call it with either 0 or the macro `EXIT_SUCCESS`. From the FDIS, 18.5.8, concerning `void exit(int status)`:

#908 Explained Report Bookmark

A character variable can store \_\_\_ character(s) at a time.

- A  
2
- B  
0
- C  
NULL
- D  
1

Correct Answer :D

## Explanation

A character variable can at a time store only one character. In fact, if we execute the following statements, what gets stored in variable ch is not really the character constant, but the ASCII value of 'A', which is 65.

```
char ch;
```

```
ch= 'A';
```

#909 [Explained](#) [Report](#) [Bookmark](#)

Which data type cannot be checked in switch-case statement?

null

- A  
enum
- B  
character
- C  
integer
- D  
float

Correct Answer :D

## Explanation

In C-languages switch/case statement is defined by the language specification to use an int value therefore we cannot use a float value in switch/case statement.

#910 [Explained](#) [Report](#) [Bookmark](#)

How many times "CCATPREPARATION" is printed?

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

```
int main()
```

```
{

 int x;

 for(x=-1; x<=10; x++)

 {

 if(x < 5)

 continue;

 else

 break;

 printf("CCATPREPARATION");

 }

 return 0;
}
```

- A  
10 times
- B  
11 times
- C  
0 times
- D  
Infinite times

Correct Answer :C

# Explanation

In program the x is initialized with -1. As  $x < 5$  (since x is -1) it will start with continue statement.

Continue means "stop the current iteration and proceed to the next iteration". Therefore x becomes 0 now. This will take place until x becomes 5.

Now if the value of x=5, it will enter the else part where it encounters the break statement, as a result it will come out of for loop. Hence it will not go to printf statement.

Therefore CCATPREPARATION will be printed 0 times.

#911 [Explained](#) [Report](#) [Bookmark](#)

How many times while loop is executed if a short int is 2 byte wide?

```
#include<stdio.h>

int main()
{
 int i=1;

 while(i <= 155)

 {
 printf("%c %d\n", i, i);
 }
}
```

```
i++;

}

return 0;
}
```

- A 153
- B 154
- C 155
- D Infinite

Correct Answer :C

## Explanation

The size of short int which is 2 byte wide does not affect the while() loop operation.

Therefore the while ( $i \leq 155$ ) loop will executed 155 times.

#912 [Explained](#) [Report](#) [Bookmark](#)

Which statement is correct about the below program?

```
#include<stdio.h>

int main()
```

```
{

 int i = 8, j = 24;

 if(i = 8) && if(j = 24)

 printf("Welcome Programmer");

 return 0;

}
```

- A Welcome Programmer
- B Error: Undeclared identifier if
- C Error: Expression syntax
- D No output

Correct Answer :C

## Explanation

In program i.e. `if(i = 8) && if(j = 24)` the "Expression syntax" error occur.

Hence the statement should be like `if((i == 5) && (j == 10))`.

Therefore on compiling the program Error: Expression syntax is occurring.

#913 [Explained](#) [Report](#) [Bookmark](#)

**Find out the error, if any in the below program?**

```
#include<stdio.h>

int main()
{
 int j = 1;

 switch(j)
 {
 printf("Hello programmer!");

 case 1:
 printf("Case1");
 break;

 case 2:
 printf("Case2");
 break;
 }

 return 0;
}
```

- A  
No error in program and prints "Case1"
- B  
Error: Invalid printf statement after switch statement
- C  
Error: No default specified
- D  
None of the above

Correct Answer :A

## Explanation

In program switch statement is used for switch(j) it becomes switch(1) because i is initialized with 1.

Therefore the case 1: block is get executed. Hence it prints "Case1".

Printf ("Hello programmer!"); is ignored by the compiler.

Hence there is no error in program and prints "Case1".

#914 [Explained](#) [Report](#) [Bookmark](#)

Find out the error, if any in the while loop of below program.

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

```
int main()
```

```
{
```

```
 int j=1;
```

```
while()

{

 printf("%d\n", j++);

 if(j>5)

 break;

}

return 0;
}
```

- **A**  
There should be a semicolon in the while statement
- **B**  
The while loop should be replaced with do-while loop
- **C**  
There should be a condition in the while loop
- **D**  
No error

Correct Answer :C

## Explanation

In program "Expression syntax" error occur because the while() loop must have conditional expression.

For Example: `while (j >5) { ... }`

**Therefore for removing the ?Expression syntax? error there should be a condition in the while loop.**

#915 [Explained](#) [Report](#) [Bookmark](#)

Find out whether both the loops in a program prints the correct string length?

```
#include<stdio.h>

main()
{
 int j;

 char s[] = "javaTpoint";

 for(j=0; s[j]; ++j)
 printf("%d \n", j);

 j=0;

 while(s[j++]);
 printf("%d ", j);

}
```

- A Compile error in the program
- B Yes, both the loops prints the correct string length
- C Only while loop prints the correct string length
- D Only for loop prints the correct string length

Correct Answer :D

## Explanation

First point every loop has terminated by a semicolon in end

Output: 10

11

In the while loop the incorrect string length is printed because while loop variable 'i' gets incremented after checking for '\0', hence giving 1 more than the length of string.

Therefore only for loop prints the correct string length.

#916 [Explained](#) [Report](#) [Bookmark](#)

Find out the error, if any in the below program?

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

```
int main()
```

```
{\n\n int P = 10;\n\n switch(P)\n {\n\n case 10:\n\n printf("Case 1");\n\n case 20:\n\n printf("Case 2");\n\n break;\n\n case P:\n\n printf("Case 2");\n\n break;\n }\n\n return 0;\n}
```

- A  
Error: Constant expression required at line case P:
- B  
Error: There is no break statement in each case
- C  
Error: No default value is specified
- D  
No error

Correct Answer :A

## Explanation

On compiling the program compiler will report an error "constant expression required" in the line case P: because variable name is not allowed to be used with case statements.

The case statements only accept constant expression. Therefore the Error: Constant expression required at line case P: is occur.

#917 [Explained](#) [Report](#) [Bookmark](#)

Find out the error, if any in the below program?

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

```
int main()
```

```
{
```

```
 int i = 1;
```

```
switch(i)

{

 case 1:

 printf("Case1");

 break;

 case 1*2+2:

 printf("Case2");

 break;

}

return 0;

}
```

- A      **Error: in switch statement**
- B      **Error: in case 1\*2+4 statement**
- C      **Error: No default specified**
- D      **No Error**

Correct Answer :D

## Explanation

In switch statement constant expression are allowed therefore in case  $1*2+4$  statement it will give no error.

Therefore it prints "Case1" in the output of program.

#918 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about below C-program?

```
#include<stdio.h>

int main()

{
 int x = 100, y = 100%80, j;

 for(j=1; j<10; j++)

 if(x != y);

 printf("x = %d y = %d\n", x, y);

 return 0;
}
```

1. The program produce the output x=100 y=20
2. The printf() function run for 10 times
3. The semi colon(;) after the if(x!=y) will not produce any error
4. The program will produce no output
  - A
  - 2
  - B
  - 1,3

- C  
3,4
- D  
4

Correct Answer :B

## Explanation

The statement 1 is true because x=100 and y=20 is the output of a program.

The statement 2 is false because printf() function is not inside for loop. Therefore printf statement only runs for 1 time.

The statement 3 is true because the semicolon is used for terminating a conditional statement. Therefore if(x!=y); is allowed in C.

The statement 4 is false because the program is producing output x=100 and y=20.

Therefore only statement 1 and 3 are correct statement.

#919 [Explained](#) [Report](#) [Bookmark](#)

Which of the following statements are correct about for loop in C-program?

1. All things that can be done using a for loop can also be done using a while loop.
2. for loop can be used if we want statements in a loop get executed at least once.
3. for loop works faster than a while loop.
4. for(;;); implements an infinite loop.

- A  
1
- B  
1,2,3
- C  
2,3,4
- D  
1,2,4

Correct Answer :D

## Explanation

**For loop is used if we want statements in loop get executed at least once. Therefore for loop works slower than a while loop i.e. statement 3 is incorrect.**

**Remaining 3 statements about for loop is correct.**

**Therefore statement 1, 2, 4 are the correct statements.**

#920 [Explained](#) [Report](#) [Bookmark](#)

**What is the output of the given program, if a short int is 2 bytes wide?**

```
#include<stdio.h>

int main()

{

 short int i = 0;
```

```

for(i<=5 && i>=-1; ++i; i>0)

 printf("%u, ", i);

return 0;

}

```

- A Expression syntax error
- B 1 .... 65535
- C 0, 1, 2, 3, 4, 5
- D No output

Correct Answer :B

## Explanation

In for loop expression i.e. `for(i<=5 && i>=-1; ++i; i>0)` the expression `i<=5 && i>=-1` is the loop condition. Expression `++i` increment the expression.

In given for loop condition the loop start from 1 and it gets executed till the limit of integer i.e. 65535.

Therefore the output of the program is 1 ... 65535.

#921 [Explained](#) [Report](#) [Bookmark](#)

Which statements are correct about an if-else statement in a C-program?

1. Nested if-else statements are allowed

2. Every if-else statement can be replaced by an equivalent statement using ?: operators
  3. Multiple statement in else block are allowed
  4. Multiple statement in if block are allowed
- A  
1 3 4
  - B  
1 2 3 4
  - C  
2 3 4
  - D  
1 4

Correct Answer :A

## Explanation

**Nested if-else statement is allowed in C-program we can use if-else statement within if or else statement.**

**Multiple statements in if or else block are allowed because we can execute multiple statements against true value of if or else condition by placing the statements within { ?.. }.**

**Mostly if-else statement can be replaced by ternary operator but there are some exceptions also in which if-else statement cannot be replaced by ternary operator.**

**Therefore 1, 3 and 4 statements are correct about if-else statement.**

#922 [Explained](#) [Report](#) [Bookmark](#)

What is the output of below C program?

```
#include<stdio.h>

int function1(int);

int main()

{

 int k=30;

 k = function1(k=function1(k=function1(k)));

 printf("k=%d\n", k);

 return 0;

}

int function1(int k)

{

 k++;

 return k;

}

• A
 k=30
• B
 k=31
```

- C  
k=32
- D  
k=33

Correct Answer :D

## Explanation

**Step 1:** int k=30; The variable k is declared as an integer type and initialized to 30.

**Step 2:** k=function1(k= function1(k=function1(k))); The function1 (k) increment the value of k by 1 and return it. In program function1(k) is called 3 times. Hence the value of k increments from k=30 to 33. Therefore result stored in the variable k=33.

**Step 3:** printf("k=%d\n", k); It prints the value of variable k =33.

#923 [Explained](#) [Report](#) [Bookmark](#)

What is the output of below program?

```
#include<stdio.h>

int j;

int function();

int main()
```

```
{

 while(j)

 {

 function();

 main();

 }

 printf("Hi\n");

 return 0;

}

int function()

{

 printf("Hello");

}
```

- A  
Hi
- B  
Hello Hi
- C  
No output
- D  
Infinite loop

Correct Answer :A

## Explanation

**Step 1:** int j; The variable j is declared as an integer type.

**Step 2:** int function(); This statement tells the compiler that the function does not accept any argument and it returns an integer value.

**Step 3:** while(j) The value of j is not initialized so the while condition is failed. Therefore it does not execute the while block.

**Step 4:** printf("Hi\n"); This statement prints "Hi".

Therefore the output of the program is "Hi".

#924 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the below program?

```
#include<stdio.h>

main() {
 int x[] = {100, 200, 300};

 printf("%d", *x +1);
}
```

- A  
10

- **B**  
200
- **C**  
101
- **D**  
201

Correct Answer :C

## Explanation

In program \*x refers to 100 and adding a 1 to \*x gives 101.

Therefore the output is 101.

#925 [Explained](#) [Report](#) [Bookmark](#)

Which of the statements are correct about 5 used in the program?

```
int num[5];

num[5]=20;
```

- **A**  
In the first statement 5 specifies an array size, whereas in the second statement it specifies a particular element of array.
- **B**  
In the first statement 5 specifies a particular element, whereas in the second statement it specifies an array size.
- **C**  
In the first statement 5 specifies a particular element, whereas in the second statement it specifies a type

- **D**  
In both the statement 5 specifies array size.

Correct Answer :A

## Explanation

The statement `int num[5];` specifies the size of array and `num[5]=20;` specifies the particular element (6th element) of the array.

Therefore in first statement 5 specifies an array size, whereas in second element it specifies a particular element of an array.

#926 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the below program?

```
#include<stdio.h>

int main()

{
 int arr[2]={20};

 printf("%d\n", 0[arr]);

 return 0;
}
```

- **A**  
2

- **B**  
0
- **C**  
20
- **D**  
16

Correct Answer :C

## Explanation

**Step 1:** int arr[2]={20}; The variable arr[2] is declared as an integer array with size of '3' and it's first element is initialized with value '20'(means arr[0]=20)

**Step 2:** printf("%d\n", 0[arr]); It prints the first element value of variable 'arr'.

Therefore the output of the program is 20.

#927 [Explained](#) [Report](#) [Bookmark](#)

In a structure, if a variable works as a pointer then from the given below operators which operator is used for accessing data of the structure using the variable pointer?

null

- **A**  
%
- **B**  
->
- **C**  
. (dot)

- D  
#

Correct Answer :B

## Explanation

**For a structure, Arrow ( ->) is used for access the data using pointer variable and Dot(.) operator can be used for accessing the data using normal structure variable.**

#928 [Explained](#) [Report](#) [Bookmark](#)

Select the correct statement which is a combination of these two statements,

Statement 1: p= (char\*) malloc(100) ;

Statement 2: char \*p;

- A  
char \*p = (char\*)malloc(100);
- B  
char \*p = (char) malloc(100);
- C  
char p = \*malloc(100);
- D  
None of the above

Correct Answer :A

## Explanation

The below code is a prototype of malloc() function, here ptr indicates the pointer.

```
ptr = (data type *)malloc(size);
```

In below code, "\*p" is a pointer of data type char and malloc() function is used for allocating the memory for char

```
char *p = (char*)malloc(100);
```

#929 [Explained](#) [Report](#) [Bookmark](#)

For the below mention C statement, what is your comment?

```
signed int *p=(int*)malloc(sizeof(unsigned int));
```

- A Would throw Runtime error
- B Improper typecasting
- C Memory will be allocated but cannot hold an int value in the memory
- D No problem with the statement

Correct Answer :D

## Explanation

The size of int and unsigned data type is same therefore there is no problem in a C statement:

```
signed int *p=(int*)malloc(sizeof(unsigned int));
```

#930 [Explained](#) [Report](#) [Bookmark](#)

**How many bytes are occupied by far, near and huge pointers in DOS (Disk Operating System)?**

- **A**  
**near=4 far=8 huge=8**
- **B**  
**near=4 far=4 huge=8**
- **C**  
**near=2 far=4 huge=4**
- **D**  
**near=2 far=4 huge=8**

**Correct Answer :C**

## **Explanation**

**Under Linux and Windows every pointer is 4 bytes long.**

**Under DOS the value of near, far and huge pointer is:**

- **near=2**
- **far=4**
- **huge=4**

**#931**Explained Report Bookmark

**Choose the correct option for the below program.**

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

```
main()
```

```
{
```

```

int *a, **b;

printf("%u\n", sizeof(a));

printf("%u\n", sizeof(b));

}

```

- **A**  
First printf() prints the value less than the second
- **B**  
Second printf() prints the value less than the first
- **C**  
Both the printf() will print the same value
- **D**  
Error in the code

Correct Answer :C

## Explanation

**Every type of pointer variable occupies the same amount of memory irrespective of any data type.**

**Therefore both the printf() statement will print the same value.**

#932 [Explained](#) [Report](#) [Bookmark](#)

Which of the function is more appropriate for reading a multi-word string?

- **A**  
`puts()`
- **B**  
`gets()`

- **C**  
printf()
- **D**  
scanf()

Correct Answer :B

## Explanation

The function gets() is used for collecting a string of characters terminated by new line from the standard input stream stdin.

Therefore gets() is more appropriate for reading a multi-word string.

#933 [Explained](#) [Report](#) [Bookmark](#)

What is the value return by strcmp() function when two strings are the same?

null

- **A**  
2
- **B**  
1
- **C**  
0
- **D**  
Error

Correct Answer :C

## Explanation

**C library function strcmp() compares the two strings with each other and the value is return accordingly.**

**int strcmp (const char \*str1, const char \*str2)**

**Comparison occurs between a first string (str1) with a second string (str2).**

**On comparing the two string, the values return by a function strcmp() are:**

- If, str1 is equal to str2 then Return value = 0
- If, str1 is greater than str2 then Return value > 0
- If, str1 is less than str2 then Return value < 0

**#934** Explained Report Bookmark

**Which of the statements are correct about the below program?**

```
#include<stdio.h>

int main()

{

 char stri[20], *p;

 printf("Enter the string\n:");

 scanf("%s", stri);

 p=stri;
```

```

while(*p != '\0')

{
 if(*p >= 97 && *p <= 122)

 *p = *p-32;

 p++;

}

printf("%s",stri);

return 0;
}

```

- A The code converts lower case character to upper case
- B The code converts upper case character to lower case
- C The code converts a string to an integer
- D Compile Time error

**Correct Answer :A**

## Explanation

**The program converts the enter string to upper case string.**

#935 [Explained](#) [Report](#) [Bookmark](#)

**What will be the output of the below program?**

```
#include<stdio.h>

main()
{
 char x[] = "Hi\0Hello";
 printf("%d %d", strlen(x), sizeof(x));
}
```

- A  
5 9
- B  
9 20
- C  
2 9
- D  
2 5

**Correct Answer :C**

## **Explanation**

**The `strlen(x)` function is used for finding the length of string 'x'. In program the length of string is count of character upto '\0'. Hence the string length output is 2.**

**The `sizeof(x)` function is used for finding the size of string 'x'. In program `sizeof()` returns the size of the complete array. Hence the size of array output is 9.**

Therefore the combined output of the program is 2 9.

#936 [Explained](#) [Report](#) [Bookmark](#)

In which stage the below code gets replaced by the contents of the file  
#include<stdio.h>

- A During linking
- B During editing
- C During preprocessing
- D During execution

Correct Answer :C

## Explanation

During preprocessing stage the line #include<stdio.h> with the system header file of that name gets replaced by the contents of file stdio.h.

Therefore the entire text of the file 'stdio.h' replaces with #include directive.

#937 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the below program?

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

```
#define SWAP(x, y) int t; t=x, x=y, y=t;
```

```

int main()

{

 int x=10, y=20;

 SWAP(x, y);

 printf("x = %d, y = %d\n", x, y);

 return 0;

}

```

- **A**  
x=10 , y=20
- **B**  
x=20, y=10
- **C**  
Error: Undefined symbol 't'
- **D**  
Error: Declaration not allowed in macro

Correct Answer :B

## Explanation

The macro statement **SWAP(x, y) int t; t=x, x=y, y=t;** swaps the value of given two variable.

**Step 1: int x=10, y=20; The variable x and y are declared as an integer type and initialized to 10, 20 respectively.**

**Step 2: SWAP(x, y);. Here the macro is substituted and it swaps the value to variable x and y.**

Hence the output of the program is x=20, y=10.

#938 [Explained](#) [Report](#) [Bookmark](#)

Which of the following are correctly formed #define statements in C language?

- A  
`#define CUBE(x) (X*X*X)`
- B  
`#define CUBE(X) {X*X*X}`
- C  
`#define CUBE (X) X*X*X`
- D  
`#define CUBE(X) (X)*(X)*(X)`

Correct Answer :D

## Explanation

The syntax for macro definition with argument is:

```
"#define MACRO_NAME(ARG) (ARG*ARG*ARG) "
```

- There should be no space between macro's name and its '(args)'.
- The variables used as macro's argument are case sensitive and its expansion should be same. i.e. 'x' and 'X' are different variables.
- A macro expansion should be enclosed within parenthesis '()' (do not use {} or []).

## #939 Explained Report Bookmark

What will be the output of the below program?

```
#include <stdio.h>

#define DEF

int main(){

 int j=3;

#define DEF

 printf("square of j=%d\n",j*j);

#define else

 printf("j=%d\n",j);

#define endif

 return 0;

}

- A
j=3
- B
square of j=9
- C
Compile error
- D
No output

```

Correct Answer :B

## Explanation

In program `#ifdef`, `#else` and `#endif` are preprocessor commands.

If the macro DEF is defined by a `#define` statement, then the code immediately after `#ifdef` is compiled otherwise code between `#else` and `#endif` commands gets executed. Since, DEF has been defined, `#ifdef DEF` evaluates to true hence, square of 'j' is calculated and gets printed i.e 9.

While defining the macro DEF, it is not compulsory to provide a expansion to the macro statement because in a program we are not using the value of macro expansion.

Therefore the output of the program is square of j=9.

#940 [Explained](#) [Report](#) [Bookmark](#)

Select the invalid predefined macro as per ANSI C.

- A `_DATE_`
- B `_TIME_`
- C `_C++_`
- D `_FILE_`

Correct Answer :C

# Explanation

There is no such macro defined with the name `_C++_`, but the `_cplusplus_` is predefined macro as per ANSI C specification.

Therefore the `_C++_` is invalid predefined macro as per ANSI C.

#941 Explained Report Bookmark

Find out the error in the below program?

```
#include<stdio.h>

int main()
{
 int j;
 #if A
 printf("Enter the number:");
 scanf("%d", &j);
 #elif B
 printf("The number is even");
 #endif
 return 0;
}
```

}

- A The number is even
- B Error: unexpected end of file because there is no matching #endif
- C Garbage values
- D None of the above

Correct Answer :B

## Explanation

The conditional macro statement #if must have an #endif statement. In the program there is no #endif statement used.

Therefore the program returns an Error: unexpected end of file because there is no matching #endif.

#942 [Explained](#) [Report](#) [Bookmark](#)

What will be the output of the below program?

```
#include<stdio.h>

#define MAX(x, y) (x > y ? x : y)
```

```
int main()

{
 int a;

 a = MAX(3+1, 2+4);

 printf("%d\n", a);

 return 0;
}
```

- A  
5
- B  
6
- C  
8
- D  
9

Correct Answer :B

## Explanation

The macro **MAX(x, y)** ( $x > y ? x : y$ ) returns the biggest value of the given two numbers.

**Step 1:** int a; The variable 'a' is declared as an integer type.

**Step 2:** a = MAX(3+1, 2+4); becomes,

=> a = (3+1 > 2+4 ? 3+1 : 2+4)

=> a = (4 > 6 ? 4 : 6)

=> a = 6

**Step 3:** printf("%d\n", a); It prints the value of variable 'a'.

Hence the output of the program is 6.

#943 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following program .

```
void main()
```

```
{
```

```
 int i=01289;
```

```
 printf("%d",i);
```

```
}
```

- A  
73
- B  
1289
- C  
0289
- D  
Error

**Correct Answer :D**

# Explanation

The prefix 0 in an integer value indicates octal value. In octal value use of 8 and 9 is not allowed and hence the error .

#944 [Explained](#) [Report](#) [Bookmark](#)

Find the output of the following program.

```
#include <stdio.h>

void main()
{
 int i=065 , j= 65;

 printf("%d %d",i,j);
}
```

- A  
065 65
- B  
53 65
- C  
65 65
- D  
053 65

Correct Answer :B

## **Explanation**

**As octal 65 ( 065 ) is equivalent of decimal value 53.**