**Answer Sheet**

1. B

Explanation

The variable declared inside the inner block replaces the x declared in the outer block, hence it prints 20 at 1st printf. When the inner block ends, the scope of inner x also ends and hence the value of x becomes 40 in the outer block.

2. A

3. A

Explanation

RHS is evaluated first and then the value is promoted to float, float arithmetic is not used here as there is no variable on the RHS that is float.The Float is printed with decimal followed by 6 zero’s.

4. C

Explanation

-3 - -3= -3 + 3 = 0

-3 --(-3)= -3 - 3 = -6

5. A

6. A

Explanation

A character variable is of 1 byte length and can store just 1 character at a time.

7. B

Explanation

The ascii value of a character constant is stored in a character variable, hence character variable is also sometimes called as a variation of int datatype.

8. A

Explanation

near=2, far=4 and huge=4 pointers exist only under DOS. Under windows and Linux every pointers is 4 bytes long.

9. C

Explanation

The default type for decimal constants is double and not float.

Hence the value 5.2 is treated as a double and 8 is printed.

10. B

Explanation

On a 32 bit compiler : 4 4 5

On a 16 bit compiler : 2 2 5

sizeof(Expr) operator returns the size of the final value of the expression.

Consider the following expression:

!d = !5.2 =0

0 is int type integer constant and it's size is 4 on a 32 bit compiler.

Now, Consider the following expression:

i = 15/2

=> i = 7

=> 7

7 is an integer and hence the size is 4 bytes.

Here the value of "i" changes only inside the scope of sizeof operator and the original i is not altered outside.

So value of variable "i" remains 5 in the printf statement.

11. D

Explanation

Modulus operator can NOT be used on FLOAT datatype

12. C

Explanation

fmod(x,y) - Calculates x modulo y, the remainder of x/y.

This function is the same as the modulus operator. But fmod() performs floating point divisions.

13. B

Explanation

External Linkage-> means global, non-static variables and functions.

Internal Linkage-> means static variables and functions with file scope.

None Linkage-> means Local variables.

14. D

Explanation

Variable names in C are made up of letters (upper and lower case) and digits. The underscore character ("\_") is also permitted. Names must not begin with a digit.

15. 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.

16. A,D

Explanation

double pow(double,double); /\* Declaration \*/

float squar(float a){} /\*Definition\*/

float b; /\*Definition\*/

extern int a; /\*Declaration\*/

Before using any variable or function it should be declared. Declaration and definition can be in single statement.

17. A

Explanation

When Program1.c is included in Program2.c, So we can say that all content of Program1.c is copied to Program2.c before compilation.

Now int x is available to program2.c so no need of extern int x; statement. it is required when in Program2.c , Program1.c is not included.

18. A

19. D

Explanation

extern x is a declaration, not definition. Hence error!

During compilation when compiler sees extern int x, it understand that int x is defined somewhere and will be available at the time of linking.

But while linking x definition is not available so it throws undeclared error.

Suppose above code is in file 1.c

Below command will give no error because it is just the compilation.

gcc -c 1.c

But gcc 1.c will throw error as it is compiling and linking. To correct the error in above program define x in a file and compile together.

/\*header.c\*/

int x;

Now, gcc 1.c header.c

and it compiles and links with no error and generate executable( a.out)

20. C

Explanation

In program A during compilation when compiler reaches to printf statement x is not known so it throws error undeclared x.

In program B during compilation first extern int x is executed and now comiler knows that x is defined somewhere, so it compiles.

In linking phase int x; is used. x is a global variable and by default it is initialized to zero.

21. A

Explanation

Any Relational expression is evaluated in terms of TRUE or FALSE. TRUE is 1 and FALSE is 0.

In this case a>6 => 8>6 is TRUE, Hence c is 1.

d=a>2 && b==3

=> 8 >2 && 4==3

=> FALSE && 4==3 (Short Circuiting does not happen since first part is FALSE in this expression.)

=> FALSE && FALSE

=> 0 && 0 => 0.

Hence d= 0

22. C

Explanation

Since i is equal to 10, the control directly jumps to the case 10 and executes:

**printf(" i=10");**

But after executing this line the compiler does not encounter **break** command hence the

case 11 is also executed i.e.

**printf(" i=11");**

After executing this line the compiler encounters break; and the control comes out of the switch block.

**Point to remember:** If a**break** statement is not encountered in a case block all the case blocks following the current case block are also executed serially until a break statement is found.

23. C

Explanation

We know that in a pre-increment operator the value is incremented first and then used, while

in post-increment operator the current value is used first and then incremented.

Now,

In the first loop,

The incremented value is compared with 10 and is printed.

initially i is 1, the control goes to the condition ( ++i < 10) in while loop, the value of i is incremented and i becomes 2 and 2< 10 is TRUE,

So, the incremented value 2 is printed. Please Note that 1 is not printed here as the value of i has already become 2 at the time of printing.

Similarly,

When the value of i is 2 , 3 is printed.

value of i     printed value

        1-----------2

        2------------3

        3------------4

        4------------5

        .  ------------ .

        .  ------------ .

        8------------9

        9------------The loop exits as ++i becomes 10.

So, 10 is not printed.

In the Second loop,

The current value of j is compared with 10 and the incremented value is printed.

So, initially j=1 and  j<10 is TRUE , Hence the printf statement is executed, by this time the value of j has become 2.

Hence 2 is printed.

value of j     printed value

        1------------2

        2------------3

        3------------4

        4------------5

        .  ----------  .

        .  ----------  .

        8------------9

        9------------10

        10------------ Loop Exits,

Since ++j is 11 which is greater than 10, the condition fails and the control comes out of the loop.

*NOTE: The statement printf("\n"); is not inside the while loop.*

24. B

Explanation

Inside the body of  while loop, as there is no parenthesis ,there is only one statement i.e. the printf statement.

The statement c++ lies outside the  while loop and is never executed.

Correct program would be:

|  |
| --- |
| while(c <100)      {          printf("%d ",c);          c++;      } |

**output:** 1 2 3 4 . . . 99

If the c++ is placed before printf statement as shown below:

|  |
| --- |
| while(c <100)      {          c++;          printf("%d ",c);      } |

The output will be: 2 3 4 . . . 99 100

Because, when c is 1 , it increments value of c to 2 and then prints 2, and so on .

when c becomes 98 -> 99 is printed and when c is 99 -> 100 is printed.

c is now 100 and  the control comes out of the loop

25. A

Explanation

Consider the while condition: (i+1? --i:j++)

It makes use of ternary operator.

i=2, j=2 initially.

i+1 = 3 which being a Non Zero Value is TRUE.

So --i is executed and  i becomes 2-1 = 1. This is the actual value that decides whether the while condition is TRUE or FALSE.

since --i = 1 is a Non Zero Value ,

**i=1 j=2 is printed.**

In the Second Iteration

i=1 and j=2.

In the condition (i+1? --i:j++)

i+1 = 2 is TRUE and the value of i is 1 , --i =0 which is FALSE,

So the while condition Fails and nothing is printed.

Hence, option A is the correct Answer.

26. B

Explanation

The Initialization and increment section is blank here, but the condition section itself increments the value of i.

Initially, i =1 and i++ being post-increment is checked if it is Equal to 0 or not before incrementing the value.

The condition statement can be written as ( i++ != 0)

So 1++ in condition prints 2,

2++ prints 3 and so on.

at one point the value of i is 32767 and is incremented by 1 , it becomes -32768 as i is a signed integer.

The condition is still satisfied , i keeps on incrementing by 1 until it becomes equal to -1.

-1++ prints 0 and i becomes 0.

This time the condition FAILS and the control comes out of the loop.

If it is run on a 16 bit compiler, It will print :

**2 3 4 5 ... 32767 -32768 -32767 ... -1 0.**

27. B

Explanation

Here, the expression a>b>c is evaluated as:

**(a>b)>c**

=**(6>5)>4** which becomes **1>4** which is **FALSE**.

Hence the **output is 0.**

Now, a few lines about Ternary operator(?:)

Let us consider an expression

Result=(W>X) ? Y : Z,

If the expression (W>X)is TRUE then the Result = Y else Result = Z.

28. B

Explanation

The precedence of + is more than << (left shift operator).

Consider the expression:  a=a+2<<1;

Hence , the above expression is (a+2)<<1

4<<1

i.e. 8  (Left Shift Doubles the value)

Now, a=8

In the second statement: a=a+(2<<1)

The Left shift operator executes before, So a=a+4= 8+4=12

Hence the answer is 8 12

29. C

Explanation

Both Line 1 and Line 2 are macro definitions. They comprise of 'macro template' MAX and SQUARE(x) and their corresponding 'macro expansion', 2 and (x\*x).

During preprocessing before compilation , the macro templates are replaced by their corresponding macro expansions.

So in above program  line 5 before compilation is =>  int y = 2 \* (4\*4);

30. D,E

Explanation

Points to remember

1. A pointer is a variable that stores the address of other variable.

    EX. int a, \*ptr; // 'ptr' is a pointer variable

    a=20;

    ptr=&a;    //Storing address of int variable 'a' in ptr pointer

Now a and ptr are pointing to same memory location.

2. \*p means value at address at which p is pointing.

     &a means address of variable a.

Option A :int \*ptr=a stores value of a in ptr which is not correct, it shoud be => int \*ptr=&a;

Option B :  ptr=a; storing value of a, which is not an address. it should be => ptr = &a;

Option C :  \*ptr = &a; here we are trying to store address of a at address where ptr is pointing, but ptr is pointing no where. it should be => ptr = &a;

31. A

Explanation

Here the precedence of ++ is higher than \* so pointer p will be incremented.

32. C

Explanation

Method

**Rule 1.** Start reading from variable,

**Rule 2.** Then read right to the variable till end

**Rule 3.**When done with right side, start with left side till end

Note: if any thing is there in (), finish it first using rule 1,2 and 3.

So float \*(\*ptr)[10];

Step 1: first (\*ptr), nothing is on right side inside bracket so move left

 ptr is a pointer

Step 2: (\*ptr)[], right side first

 ptr is a pointer to an array of

Step 3: \*(\*ptr)[], now left side

 ptr is a pointer to an array of of pointer to

Step 4: float \*(\*ptr)[]

 ptr is a pointer to an array of of pointer to float type

33. A

Explanation

NULL is #defined as 0 in the '**stdio.h'** file. So both p and t are null pointers.

34. C

Explanation

char \*ptr=str, ptr is pointing to the first element of str array.

\*ptr='H', this will store 'H' to the first position of str array.

35. A

Explanation

Compiler by default assumes return type of undeclared function to int and accept unspecified number of argument.

Here **displayOut** is called before it is defined. In this case the compiler assumes that the function dispayOut() prototype is declared as int displayOut(); but when in definition compiler sees that it is returning int which it has assumed so no error and warning.

See below example

|  |
| --- |
| /\* program1.c\*/  #include<stdio.h>  int main()  {     displayOut();     return 0;  }    void dispalyOut()  {   printf("Hi from placementyogi.com\n");  } |

In above program( program1.c) return type of displayOut() is void but the datatype assumed by compiler is int so conflicting data type warning.

36. C

Explanation

Yes, it's true. Definition of a function is not allowed inside body of any other function in Standard C .

If you are using **gcc** compiler then program compiles.

37. B

Explanation

A stack(call stack) is used to store information about the active subroutines of a computer program. So whenever control transfer from one function(subroutine) to another function, return address and other parameter are stored on stack.

In above program main is calling itself repeatedly, it creates an infinite loop. Each call pushes return address and other parameters to call stack and it continues till stack is full. When stack is full it terminates with error : Segmentation fault (core dumped)

38. E

Explanation

All A,B,C and D are valid declarations.

extern, auto, static and register are storage classes.

There is no keyword "global" in c and hence the declaration

global int i; is invalid.

39. B

Explanation

We are merely requesting the CPU to store the variable in the registers.

It depends on the CPU whether it really allocates it on the registers or not.

40. D

Explanation

error: address of register variable ‘a’ requested

We can not refer the address of a register variable because it is not stored in the memory but on the registers.

Only variable stores in the memory have an address.

41. A

Explanation

A static variable can be defined as many times as we wish in the global scope. The Declaration after initialization is ignored and the value of i remains 10.

42. C

Explanation

int[] a; // this is not a valid way of declaring an array.

int a[]; // this is valid

43. A

Explanation

**arr gives the address of the first element**, whereas**&arr gives the address of the whole array**.

arr+1 gives the address of the 2nd element of the array ,

i.e. **arr + sizeof(int)= 1000+4 = 1004**

&arr + 1 gives the address of the next array.

i.e. **&arr+1 = 1000 + sizeof(int)\*5** = 1000 +20 =**1020**

44.  C

Explanation

c[] is an array of characters.

p[3] = 'e' i.e. ascii value of 'e'

p[1] = 'a' i.e. ascii value of 'a'

p[3]-p[1] has to be 4, whatever be the ascii value of 'e' and 'a'.

So the expression p + p[3]-p[1] becomes p+4.

**p[4] is '2' in the string 'gate2011'**.

Hence the string is printed from this index till '\0' is encountered.

So, "**2011**" is printed.

Note: This question was asked in GATE 2011 Computer Science Paper.

45. B

Explanation

semicolon(**;**) is missing at end of structure declaration.

46. A,C,D

Explanation

malloc, calloc and realloc can be used to allocate new memory. See below for  detail usage and explanation

**malloc()** : allocates the specified number of bytes. malloc returns a void pointer (void \*)

**calloc()** : allocates the specified number of memory blocks of given size and initializes them to zero.

**realloc()** : increases,decrease and allocates the size of the specified block of memory.

47. B

Explanation

**free()** : releases the specified block of memory back to the system.

|  |
| --- |
| free(ptr); |

48. B

49.  B

Explanation

malloc() : allocates the specified number of bytes and always returns a void pointer (void \*)

50. A

51.  A

Explanation

C programming language provides a keyword called typedef which you can use to give a type a new name.

In above question myint is the new name of int type. Now we you use myint in place of int, both are same.

52. C

53. A

54. A

55. C

Explanation

fp has been declared as a pointer to structure called FILE which has been defined in header file <stdio.h>.For accessing any file we must open the file in the proper mode.

**fopen("file name","mode")**

In above question strcpy function returns the base address of string str[], into which"myfile.txt" has been copied.The same is passed to fopen(), and "myfile.txt" opened in write mode. it indicates that fopen() needs the pointer to the name of file we want it to open. **How we supply the name doesn't matter**.

56. A,D

57. A

58. D

59. B

60. A

61. B

Explanation

It’s really easy to convert hexadecimal number to binary. Each number in hexadecimal represents 4 binary bits.

Example:

                 1      2     3      4

0x1234 = 0001 0010 0011 0100

In above example 1 is converted  to 0001, 2 to 0010, 3 to 0011 and 4 to 0100 and done!!

62. C

63. A

64. B

Explanation

Nowhere other than initialization can a program assign a value to a const variable. 'a' should have been initialized when it is defined.

Ex. const int a=100;

65.A

66. C

67. C

Explanation

p is a pointer to a "constant integer". Here we are trying to change the value of the "constant integer" that is not allowed.

68. D

69. B

Explanation

Every actual argument list must be known at compile time.

70. A

71. B

72. B

73. A

74. C

Explanation

In this program the compiler will not know that the function *display()* exists. So, the compiler will generate "Type mismatch in redeclaration of function *display()*".

To over come this error, we have to add function prototype of function *display()*.  
Another way to overcome this error is to define the function *display()* before the *int main();* function.

#include<stdio.h>

void display(); /\* function prototype \*/

int main()

{

display();

return 0;

}

void display()

{

printf("India.com");

}

**Output**: India.com

75. A

Explanation

True, we can use long double; if double range is not enough.

double = 8 bytes.

long double = 10 bytes

76. A

Explanation

True, When a function is declared inside the source file, that function(local function) get a priority than the extern function. So there is no need to declare a function as extern inside the same source file.

77. B

Explanation

Given 3.14 is a double constant.

To specify 3.14 as long double, we have to add L to the 3.14. (i.e 3.14L)

78. C

Explanation

The macro "NULL" is defined in locale.h, stddef.h, stdio.h, stdlib.h, string.h, time.h, and wchar.h.

79. D

80. B

Explanation

NULL is #defined as 0 in the 'stdio.h' file. Thus, both p and t are NULL pointers.

81. B

82. B

83. B

84. B

Explanation

Hexadecimal system is better, because each 4-digit binary represents one Hexadecimal digit

85. B

86. C

87. D

88. C

89. D

Explanation

rewind() takes the file pointer to the beginning of the file. so that the next I/O operation will take place at the beginning of the file.

Example: rewind(FilePointer);

90. D

Explanation

strrchr() returns a pointer to the last occurrence of character in a string.

Example:

#include <stdio.h>

#include <string.h>

int main()

{

char str[30] = "12345678910111213";

printf("The last position of '2' is %d.\n",

strrchr(str, '2') - str);

return 0;

}

Output: The last position of '2' is 14.

91. C

Explanation

The standard error(stderr) stream is the default destination for error messages and other diagnostic warnings. Like stdout, it is usually also directed to the output device of the standard console (generally, the screen).

92. 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.

93. D

Explanation

The keyword return is used to transfer control from a function back to the calling function.

Example:

#include<stdio.h>

int add(int, int); /\* Function prototype \*/

int main()

{

int a = 4, b = 3, c;

c = add(a, b);

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

return 0;

}

int add(int a, int b)

{

/\* returns the value and control back to main() function \*/

return (a+b);

}

Output:

c = 7

94. C

Explanation

If the index of the array size is exceeded, the program will crash. Hence "option c" is the correct answer. But the modern compilers will take care of this kind of errors.

Example: Run the below program, it will crash in Windows (TurboC Compiler)

#include<stdio.h>

int main()

{

int arr[2];

arr[3]=10;

printf("%d",arr[3]);

return 0;

}

Since C is a compiler dependent language, it may give different outputs at different platforms. We have given the Turbo-C Compiler (Windows) output.

Please try the above programs in Windows (Turbo-C Compiler) and Linux (GCC Compiler), you will understand the difference better.

95. B

96. C

Explanation

The statement 'C' is correct. When we pass an array as a funtion argument, the base address of the array will be passed.

97. C

Explanation

Declaration: char \*fgets(char \*s, int n, FILE \*stream);

fgets reads characters from stream into the string s. It stops when it reads either n - 1 characters or a newline character, whichever comes first.

Therefore, the string str contain "I am a boy\n\0"

98. A

Explanation

The file source.txt will be opened in the binary mode.

99. B

Explanation

The fp is a structure which contains a char pointer which points to the first character of a file.

100. 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);