1. C accepts both lowercase and uppercase alphabets as variables and functions.
Identifier names must be different from keywords. You cannot use int as an identifier because int is a keyword.

2. C is a strongly typed language. This means that the variable type cannot be changed once it is declared. To store the decimal values in C, you need to declare its type to either double or float.

3. A character literal is created by enclosing a single character inside *single* quotation marks.
For example: 'a', 'm', 'F', '2', '}' etc.

4. \n Newline Character

5. If you want to define a variable whose value cannot be changed, you can use the const keyword. This will create a constant. For example,
const int number = 5;

6.

int	%d,
float	%f
long int	%ld
long long int	%lld

Bytes occupied by various data types:

1. Int : 4
2. Char: 1
3. Float: 4
4. Double: 8
5. Long int: 4
6. Long long int: 8

7. You can declare multiple variables at once in C programming. For example int id, age;

8. printf() is one of the main output function. The function sends formatted output to the screen. scanf() is one of the commonly used function to take input from the user. To use printf()/scanf() in our program, we need to include stdio.h header file using the #include <stdio.h> statement.

9. All valid C programs must contain the main() function. The code execution begins from the start of the main() function.

10. The return 0; statement inside the main() function is the "Exit status" of the program. It's optional.

11. printf("Number = %d", testInteger);
We use %d format specifier to print int types. Here, the %d inside the quotations will be replaced by the value of testInteger.
To print float, we use %f format specifier. Similarly, we use %lf to print double values.

12.
```
// for single line comment

/*
multi line comment
   */
```

13.

%	remainder after division (modulo division)
/	division

14. When dividing two numbers, if we want answer in decimal (for ex: 5/2 = 2.5), then either of the number being divided should be of float data type. If both are of int data type, then we will get the integer part only. Also, the variable containing the answer should also be of float data type.

15. The modulo operator % computes the remainder. When a=9 is divided by b=4, the remainder is 1. The % operator can only be used with integers.

16. Post vs Pre increment:
If you use the ++ operator as a prefix like: ++var, the value of var is incremented by 1; then it returns the value.
If you use the ++ operator as a postfix like: var++, the original value of var is returned first; then var is incremented by 1.

17.

&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

18. Bitwise operators are characters that represent actions (bitwise operations) to be performed on single bits.

19.
The syntax of the if statement in C programming is:

```
if (test expression)
{
  // code
}
```

IMPORTANT: If the body of an if/else if/else statement has only one statement, you do not need to use brackets {}.

20.
The syntax of the for loop is:

```
for (initializationStatement; testExpression; updateStatement)
{
   // statements inside the body of loop
}
```

IMPORTANT: It is worth knowing that parameters inside for() are seperated by ; and not , eg: for(i=1;i<=10;++i)

21.
C follows top-down execution.

22.
```
while(1)    //Even while(2) etc works, any number other than 0 works.
```

```
{
}
```

and

```
for( ; ; )
{
}
```

is an infinite loop

23. to print anything without using the semicolon, put the text inside the if(). It is valid and there is no error.

24. return value of a scanf function is the number of inputs it has taken from the user:

for example in the following code snippet:

```
#include<stdio.h>
int main()
{
   int n, w, m;
   printf("%d", scanf("%d %d %d", &n, &w, &m));

}
```

the output here is 3 because scanf inputted 3 values.

25. When we use the division arithmetic operator:
int/int gives int (even if you define the variable that stores the variable as a float, it will still be an integer albeit 3.000000, float/int or int/float gives a float value (Remember you have to save it as a float variable for output to be a float number)

26. Explicit Type Casting

```
int main()
{
double x =1.2;
```

```
int sum = int(sum)+1;
printf({"sum=%d", sum);
return 0;

}
```

27. Implicit Type Casting:

When you add a character to an integer then the c language takes the ASCII value of the character and adds it up to the integer.
implicit type casting converts the data type of lower bit value to the bit value of the data type with the higher bit value.
for example, a character just occupies 1 byte while an integer occupies 4 bytes. so a character is converted to the integer data type (through the ASCII value)

28. do/while loop:

The do/while loop is similar to the while loop with one important difference. The body of do...while loop is executed at least once. Only then, the test expression is evaluated.

```
do {
  // the body of the loop
}
while (testExpression);
```

first the body of the loop will be executed, and then control will go to the while. if testExpression is true then control will again go to the body of the loop or else if it is false then the loop breaks. either way, the point is, the body of the loop will be executed at least once.
The do/while loop executes at least once i.e. the first iteration runs without checking the condition. The condition is checked only after the first iteration has been executed.

29. The break statement ends the loop immediately when it is encountered. Its syntax is: break;
The break statement is almost always used with if/else statement inside the loop.

30. The continue statement skips the current iteration of the loop and continues with the next iteration. Its syntax is: continue;
The continue statement is almost always used with the if/else statement.

31. switch/case syntax:

```
switch (expression)
{
    case constant1:
      // statements
      break;

    case constant2:
      // statements
      break;
    .
    .
    .
    default:
      // default statements
}
```

The expression is evaluated once and compared with the values of each case label.
If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to constant2, statements after case constant2: are executed until break is encountered.
If there is no match, the default statements are executed. The default clause inside the switch statement is optional.
If we do not use the break statement, all statements after the matching label are also executed.

[skipped goto label, rarely used]

Functions:

32.
The execution of a C program begins from the main() function.
When the compiler encounters functionName();, control of the program jumps to void functionName()
And, the compiler starts executing the codes inside functionName().
The control of the program jumps back to the main() function once code inside the function definition is executed.
Note, function names are identifiers and should be unique.

33. function prototype:
A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.
IMP: The function prototype is not needed if the user-defined function is defined before the main() function.
Syntax:
returnType functionName(type1 argument1, type2 argument2, ...);

34. Calling a function:
Control of the program is transferred to the user-defined function by calling it.

Syntax of function call:
functionName(argument1, argument2, ...);

35. Arguments:
Argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during the function call.
A function can also be called without passing an argument.

36. Return Statement
The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.

The type of value returned from the function and the return type specified in the function prototype and function definition must match.

37. Syntax of return statement
return (expression);

return a;
return (a+b);

Arrays:

38. An Array is a static data type, it has a fixed pre-defined memory size which cannot be

changed later, it is not dynamic in nature. Array has advantages as well as its disadvantages.

39. Size of an array is the number of bytes occupied by the aray. If an integer is stored in array, each integer occupies 4 bytes. if 5 integer are stored in an array then size of the array is 5*4=20. which is the size of the aray.

```
for(i=0;i<n;i++)
scanf("%d", &arr[i])
```

40. Arrays have 0 as the first index, not 1. Suppose the starting address of mark[0] is 2120d. Then, the address of the mark[1] will be 2124d. Similarly, the address of mark[2] will be 2128d and so on.
This is because the size of a float is 4 bytes.

41.
```
int mark[5]={1,2,3,4,5};
int mark[]={1,2,3,4,5}
```
You can skip writing the size of the array if you are intitalizng it while defining it.Suppose the starting address of mark[0] is 2120d. Then, the address of the mark[1] will be 2124d. Similarly, the address of mark[2] will be 2128d and so on.
This is because the size of a float is 4 bytes.

42. Access elements out of its bound:
This may cause unexpected output (undefi
ned behavior). Sometimes you might get an error and some other time your program may run correctly.

43. Multi-dimensional array:
C uses a Row-Major Form.

```
float x[3][4];
```
Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.

```
// Different ways to initialize two-dimensional array

int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};

int c[][3] = {{1, 3, 0}, {-1, 5, 9}};
//In a multidimensional array, it is okay to skip the first dimension but don't skip the other
```

dimensions.

int c[2][3] = {1, 3, 0, -1, 5, 9};

44. To pass an entire array to a function, only the name of the array is passed as an argument. However, notice the use of [] in the function definition.

```
float calculateSum(float num[]) {
... ..
}
```
This informs the compiler that you are passing a one-dimensional array to the function.

45.
To pass multidimensional arrays to a function, only the name of the array is passed to the function (similar to one-dimensional arrays).
For passing a two-dimensional array, use [][] in the function definiton.
When passing two-dimensional arrays, it is not mandatory to specify the number of rows in the array. However, the number of columns should always be specified.

Pointers:

46. If you have a variable var in your program, &var will give you its address in the memory.

47. Pointers (pointer variables) are special variables that are used to store addresses rather than values.

48.
int* p;
Here, we have declared a pointer p of int type, or use:
int *p; (either way works)

49.
int* p1, p2;
Here, we have declared a pointer p1 and a normal variable p2. (p2 is not a pointer, it is a normal variable)

int* x, y
is parsed as int (*x), y
according to the C standard

50.

```
int* pc, c;
c = 5;
pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

```
*pc=2;
```

This change the value at the memory location pointed by the pointer pc to 2.

51.

```
int x[]={1,2,3,4};
```

The address of &x[0] and x is the same. It's because the variable name x points to the first element of the array and so does &x[0]
Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

52. In C programming, it is also possible to pass addresses as arguments to functions.
To accept these addresses in the function definition, we can use pointers. It's because pointers are used to store addresses.

53.

```
#include <stdio.h>
void swap(int* n1, int* n2)
{
   int temp;
   temp = *n1;
   *n1 = *n2;
   *n2 = temp;
}
//use the void function whenever you don't want to return anything or just want to use the function to print something.

int main()
{
   int num1 = 5, num2 = 10;
```

```
    // address of num1 and num2 is passed
    swap( &num1, &num2);

    printf("num1 = %d\n", num1);
    printf("num2 = %d", num2);
    return 0;
}
```

Dynamic Memory Allocation (Malloc)

54.

The name "malloc" stands for memory allocation.

The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be typecasted into pointers of any data type.

```
ptr = (castType*) malloc(size);
```

Example

```
ptr = (float*) malloc(100 * sizeof(float));
```

55. Dynamically allocated memory created with malloc() doesn't get freed on their own. You must explicitly use free() to release the space.

```
free(ptr);
```

This statement frees the space allocated in the memory pointed by ptr.

```
// Program to calculate the sum of n numbers entered by the user

#include <stdio.h>
#include <stdlib.h>

int main() {
  int n, i, *ptr, sum = 0;

  printf("Enter number of elements: ");
  scanf("%d", &n);

  ptr = (int*) malloc(n * sizeof(int));
```

```
 // if memory cannot be allocated
 if(ptr == NULL) {
   printf("Error! memory not allocated.");
   exit(0);
 }

 printf("Enter elements: ");
 for(i = 0; i < n; ++i) {
   scanf("%d", ptr + i);
   sum += *(ptr + i);
 }

 printf("Sum = %d", sum);

 // deallocating the memory
 free(ptr);

 return 0;
}
```

Dynamically allocating an array of pointers follows the same rule as arrays of any type:

```
type *p;

 p = malloc(m* sizeof *p);
```

Strings in C:

A string is a sequence of characters terminated with a null character \0
example: char c[] = "c string";
When the compiler encounters a sequence of characters enclosed in the double quotation marks (not single quotation marks), it appends a null character \0 at the end by default.

You can initialize strings in a number of ways.

```
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = "abcde"; //Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters. This is bad and you should never do this.
```

You always have to assign a string at the same time while declaring it, not after it. Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared. For example,

```
char c[100];
c = "C programming";  // Error! array type is not assignable.
```

You can use the scanf() function to read a string. The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.). The format specifier for strings is %s

Example: scanf() to read a string

```
#include <stdio.h>
int main()
{
   char name[20];
   printf("Enter name: ");
   scanf("%s", name);
   printf("Your name is %s.", name);
   return 0;
}
```

Output:

```
Enter name: Dennis Ritchie
Your name is Dennis.
```

Notice that we have used the code name instead of &name with scanf().

```
// scanf("%s", name);
```

This is because name is a char array, and we know that array names decay to pointers in C. Thus, the name in scanf() already points to the address of the first element in the string, which is why we don't need to use &.

You can use the fgets() function to read a line of string. And, you can use puts() to display the string.

Example: fgets() and puts()

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

```
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Strings can be passed to a function in a similar way as arrays, because both of them are a pointer.

```
#include <stdio.h>

int main(void) {
  char name[] = "Harry Potter";

  printf("%c", *name);     // Output: H
  printf("%c", *(name+1));   // Output: a
  printf("%c", *(name+7));   // Output: o

  char *namePtr;

  namePtr = name;
  printf("%c", *namePtr);     // Output: H
  printf("%c", *(namePtr+1));   // Output: a
  printf("%c", *(namePtr+7));   // Output: o
}
```

C supports a large number of string handling functions in the standard library "string.h".
Note: Though, gets() and puts() function handle strings, both these functions are defined in "stdio.h" header file itself.

String Functions:

1) C strlen() : The strlen() function calculates the length of a given string.

The strlen() function takes a string as an argument and returns its length. The returned value is of type size_t (an unsigned integer type).

```
// use the %zu format specifier to print size_t
printf("Length of string a = %zu \n",strlen(a));
```

Note that the strlen() function doesn't count the null character \0 while calculating the length.

2) C strcpy(): The strcpy() function copies and returns the string pointed by source (including the null character) to the destination (a already intialized empty string).

The function prototype of strcpy() is: char* strcpy(char* destination, const char* source);

Example:

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

int main() {
  char str1[20] = "C programming";
  char str2[20];

  strcpy(str2, str1);   // copying str1 to str2
  puts(str2); // C programming
  return 0;
```

Note: When you use strcpy(), the size of the destination string should be large enough to store the copied string. Otherwise, it may result in undefined behaviour.

3) C strcat(): The function definition of strcat() is:
char *strcat(char *destination, const char *source)

The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[100] = "Hello ", str2[] = "World";

   // concatenates str1 and str2
   // the resultant string is stored in str1.
   strcat(str1, str2);

   puts(str1); //Output: Hello World
   puts(str2); //Output: World
   return 0;
}
```

Note: When we use strcat(), the size of the destination string should be large enough to store the resultant string. If not, we will get the segmentation fault error.

4) C strcmp(): The strcmp() compares two strings character by character. The function prototype of strcmp() is:

```
int strcmp (const char* str1, const char* s
```

Return Value	Remarks
0	if strings are equal
>0	if the first non-matching character in str1 is greater (in ASCII) than that of str2.
<0	if the first non-matching character in str1 is lower (in ASCII) than that of str2.

Example: C strcmp() function

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

int main() {
  char str1[] = "abcd", str2[] = "abCd", str3[] = "abcd";
  int result;

  result = strcmp(str1, str2);  // comparing strings str1 and str2
  printf("strcmp(str1, str2) = %d\n", result);

  result = strcmp(str1, str3);   // comparing strings str1 and str3
  printf("strcmp(str1, str3) = %d\n", result);

  return 0;
}
```

Output:

```
strcmp(str1, str2) = 1
strcmp(str1, str3) = 0
```