Q.1

a. t’s a pointer variable which can hold the address of another pointer variable. It de-refers twice to point to the data held by the designated pointer variable.

Eg: int x = 5, \*p=&x, \*\*q=&p;

Therefore ‘x’ can be accessed by \*\*q.

b. Used to resolve the scope of global symbol.

Eg:

main() {

extern int i;

Printf(“%d”,i);

}

int i = 20;

c. Preprocessor is a directive to the compiler to perform certain things before the actual compilation process begins.

d. The arguments which we pass to the main() function while executing the program are called as command line arguments. The parameters are always strings held in the second argument (below in args) of the function which is array of character pointers. First argument represents the count of arguments (below in count) and updated automatically by operating system.

main( int count, char \*args[]) {

}

e. sizeof

Q.2

A free().

b. Declaration associates type to the variable whereas definition gives the value to the variable.

q.3

a. Difference Between break and continue

|  |  |
| --- | --- |
| Difference Between break and continue | |
| break | continue |
| A break can appear in both switch and loop (for, while, do) statements. | A continue can appear only in loop (for,while, do) statements. |
| A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered. | A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue. |
| The break statement can be used in both switch and loop statements. | The continue statement can appear only in loops. You will get an error if this appears in switch statement. |
| When a break statement is encountered, it terminates the block and gets the control out of the switch or loop. | When a continue statement is encountered, it gets the control to the next iteration of the loop. |
| A break causes the innermost enclosing loop or switch to be exited immediately. | A continue inside a loop nested within aswitch causes the next loop iteration. |

b. C malloc()

The name malloc stands for "memory allocation".

The function malloc() reserves a block of memory of specified size and return a [pointer](https://www.programiz.com/c-programming/c-pointers) of type void which can be casted into pointer of any form.

Syntax of malloc()

ptr = (cast-type\*) malloc(byte-size)

Here, ptr is pointer of cast-type. The malloc() function returns a pointer to an area of memory with size of byte size. If the space is insufficient, allocation fails and returns NULL pointer.

ptr = (int\*) malloc(100 \* sizeof(int));

This statement will allocate either 200 or 400 according to size of int 2 or 4 bytes respectively and the pointer points to the address of first byte of memory.

C calloc()

The name calloc stands for "contiguous allocation".

The only difference between malloc() and calloc() is that, malloc() allocates single block of memory whereas calloc() allocates multiple blocks of memory each of same size and sets all bytes to zero.

Syntax of calloc()

ptr = (cast-type\*)calloc(n, element-size);

This statement will allocate contiguous space in memory for an array of nelements. For example:

ptr = (float\*) calloc(25, sizeof(float));

This statement allocates contiguous space in memory for an array of 25 elements each of size of float, i.e, 4 bytes.

C free()

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

syntax of free()

free(ptr);

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

Example #1: Using C malloc() and free()

Write a C program to find sum of n elements entered by user. To perform this program, allocate memory dynamically using malloc() function.

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num, i, \*ptr, sum = 0;

printf("Enter number of elements: ");

scanf("%d", &num);

ptr = (int\*) malloc(num \* sizeof(int)); //memory allocated using malloc

if(ptr == NULL)

{

printf("Error! memory not allocated.");

exit(0);

}

printf("Enter elements of array: ");

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

{

scanf("%d", ptr + i);

sum += \*(ptr + i);

}

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

free(ptr);

return 0;

}

Q.4

#include<stdio.h>

#include<stdlib.h>

int main(){

  int i,j,k,l,n;

system("cls");

printf("enter the range=");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

for(j=1;j<=n-i;j++)

{

printf(" ");

}

for(k=1;k<=i;k++)

{

printf("%d",k);

}

for(l=i-1;l>=1;l--)

{

printf("%d",l);

}

printf("\n");

}

return 0;

}

B

#include<stdio.h>

 int main()

{

int n,r,sum=0,temp;

printf("enter the number=");

scanf("%d",&n);

temp=n;

while(n>0)

{

r=n%10;

sum=sum+(r\*r\*r);

n=n/10;

}

if(temp==sum)

printf("armstrong  number ");

else

printf("not armstrong number");

return 0;

}