EXERCISE 6

Content:

1. Nested Loops in C
2. Infinite Loop in C
3. C break
4. C continue
5. C goto
6. Type Casting

I. Nested Loops in C

C supports nesting of loops. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define 'while' loop inside a 'for' loop.

Syntax of Nested loop

```
Outer_loop
{
    Inner_loop
   {
         // inner loop statements.
   }
       // outer loop statements.
}
```

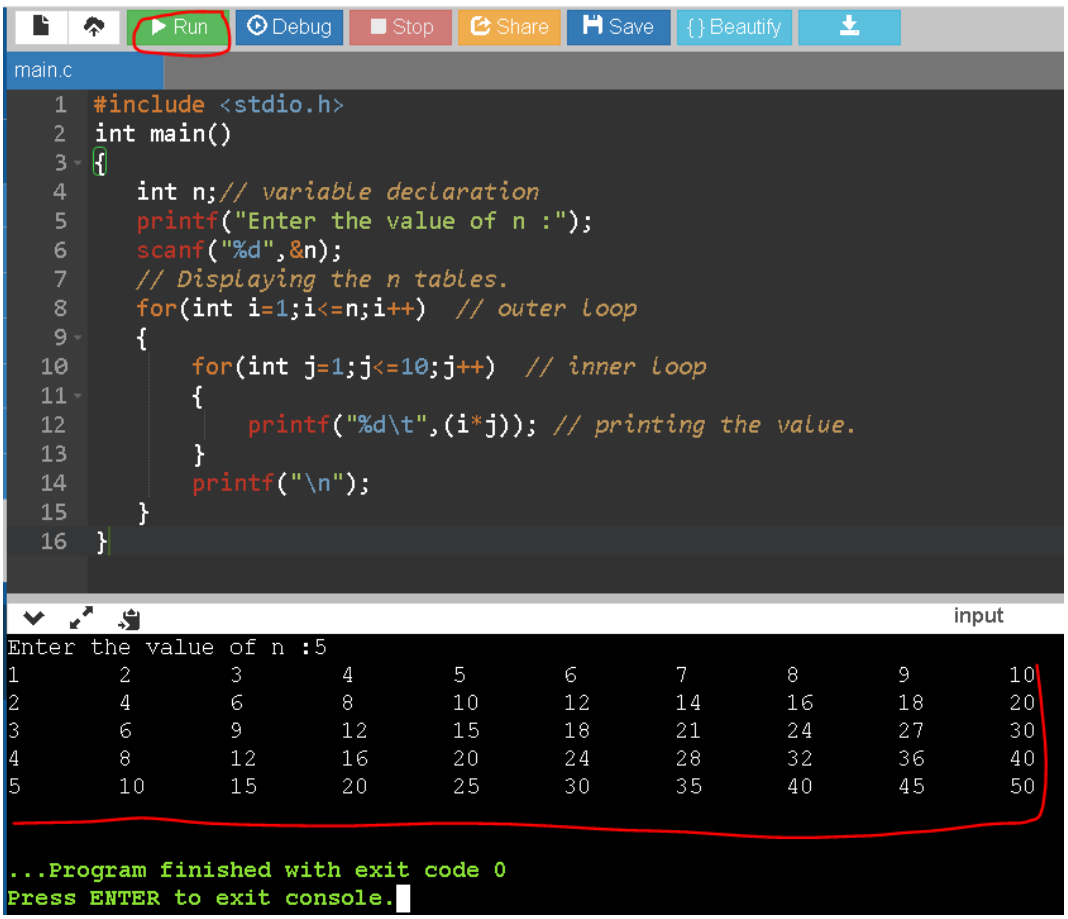
Outer_loop and Inner_loop are the valid loops that can be a 'for' loop, 'while' loop or 'do-while' loop.

Nested for loop

The nested for loop means any type of loop which is defined inside the 'for' loop.

```
for (initialization; condition; update)
{
    for(initialization; condition; update)
    {
            // inner loop statements.
    }
    // outer loop statements.
}
```

LAB: Example of nested for loop



Explanation of the above code

- First, the 'i' variable is initialized to 1 and then program control passes to the i<=n.

- The program control checks whether the condition 'i<=n' is true or not.
- If the condition is true, then the program control passes to the inner loop.
- The inner loop will get executed until the condition is true.
- After the execution of the inner loop, the control moves back to the update of the outer loop, i.e., i++.
- After incrementing the value of the loop counter, the condition is checked again, i.e., i<=n.
- If the condition is true, then the inner loop will be executed again.
- This process will continue until the condition of the outer loop is true.

Nested while loop

The nested while loop means any type of loop which is defined inside the 'while' loop.

```
while(condition)
{
    while(condition)
    {
         // inner loop statements.
    }
// outer loop statements.
}
```

LAB: Example of nested while loop:

main.c

```c
#include <stdio.h>
int main()
{
   int rows;  // variable declaration
   int columns; // variable declaration
   int k=1; // variable initialization
   printf("Enter the number of rows :");  // input the number of rows.
   scanf("%d",&rows);
   printf("\nEnter the number of columns :"); // input the number of columns.
   scanf("%d",&columns);
      int a[rows][columns]; //2d array declaration
      int i=1;
   while(i<=rows) // outer loop
   {
       int j=1;
      while(j<=columns)  // inner loop
       {
           printf("%d\t",k);  // printing the value of k.
           k++;   // increment counter
           j++;
       }
       i++;
       printf("\n");
   }
}
```

Explanation of the above code.

- We have created the 2d array, i.e., int a[rows][columns].
- The program initializes the 'i' variable by 1.
- Now, control moves to the while loop, and this loop checks whether the condition is true, then the program control moves to the inner loop.
- After the execution of the inner loop, the control moves to the update of the outer loop, i.e., i++.
- After incrementing the value of 'i', the condition (i<=rows) is checked.
- If the condition is true, the control then again moves to the inner loop.
- This process continues until the condition of the outer loop is true.

Result/Output:

```
Enter the number of rows :5

Enter the number of columns :3
1        2        3
4        5        6
7        8        9
10       11       12
13       14       15


...Program finished with exit code 0
Press ENTER to exit console.
```

Nested do..while loop

The nested do..while loop means any type of loop which is defined inside the 'do..while' loop.

```
do
{
   do
  {
      // inner loop statements.
   }while(condition);
// outer loop statements.
}while(condition);
```

LAB: Example of nested do..while loop:

```c
#include <stdio.h>
int main()
{
  /*printing the pattern
      ********
      ********
      ********
      ******** */
int i=1;
do            // outer loop
{
    int j=1;
    do        // inner loop
   {
       printf("*");
       j++;
   }while(j<=8);
    printf("\n");
    i++;
     }while(i<=4);
}
```

Result/Output:

```
********
********
********
********

...Program finished with exit code 0
Press ENTER to exit console.
```

Explanation of the above code.

- First, we initialize the outer loop counter variable, i.e., 'i' by 1.
- As we know that the do..while loop executes once without checking the condition, so the inner loop is executed without checking the condition in the outer loop.
- After the execution of the inner loop, the control moves to the update of the i++.

- When the loop counter value is incremented, the condition is checked. If the condition in the outer loop is true, then the inner loop is executed.
- This process will continue until the condition in the outer loop is true.

II. Infinite Loop in C

What is infinite loop?

An infinite loop is a looping construct that does not terminate the loop and executes the loop forever. It is also called an indefinite loop or an endless loop. It either produces a continuous output or no output.

When to use an infinite loop?

An infinite loop is useful for those applications that accept the user input and generate the output continuously until the user exits from the application manually. In the following situations, this type of loop can be used:

- All the operating systems run in an infinite loop as it does not exist after performing some task. It comes out of an infinite loop only when the user manually shuts down the system.
- All the servers run in an infinite loop as the server responds to all the client requests. It comes out of an indefinite loop only when the administrator shuts down the server manually.
- All the games also run in an infinite loop. The game will accept the user requests until the user exits from the game.

We can create an infinite loop through various loop structures. The following are the loop structures through which we will define the infinite loop:

- for loop
- while loop
- do-while loop
- go to statement
- C macros

1. For loop

Let's see the infinite 'for' loop. The following is the definition for the infinite for loop:

```
for(; ;)
{
    // body of the for loop.
```

```
}
```

As we know that all the parts of the 'for' loop are optional, and in the above for loop, we have not mentioned any condition; so, this loop will execute infinite times.

Let's understand through an example.

In the above code, we run the 'for' loop infinite times, so "**Hello!**" will be displayed infinitely.

2. while loop

Now, we will see how to create an infinite loop using a while loop. The following is the definition for the infinite while loop:

```
while(1)
{
    // body of the loop..
}
```

In the above while loop, we put '1' inside the loop condition. As we know that any non-zero integer represents the true condition while '0' represents the false condition.

Let's look at a simple example.

```
main.c
1  #include <stdio.h>
2  int main()
3  {
4    int i=0;
5    while(1)
6    {
7        i++;
8        printf("i is :%d",i);
9    }
10 return 0;
11 }
```

In the above code, we have defined a while loop, which runs infinite times as it does not contain any condition. The value of 'i' will be updated an infinite number of times.

Result/Output:

```
s :12481i is :12482i is :12483i is :12484i is :12485i is :12486i
493i is :12494i is :12495i is :12496i is :12497i is :12498i is :1
is :12506i is :12507i is :12508i is :12509i is :12510i is :12511i
2518i is :12519i is :12520i is :12521i is :12522i is :12523i is :
 is :12531i is :12532i is :12533i is :12534i is :12535i is :12536
12543i is :12544i is :12545i is :12546i is :12547i is :12548i is
```

3. do..while loop

The do..while loop can also be used to create the infinite loop. The following is the syntax to create the infinite do..while loop.

```
do
{
    // body of the loop..
}while(1);
```

The above do..while loop represents the infinite condition as we provide the '1' value inside the loop condition. As we already know that non-zero integer represents the true condition, so this loop will run infinite times.

4. goto statement

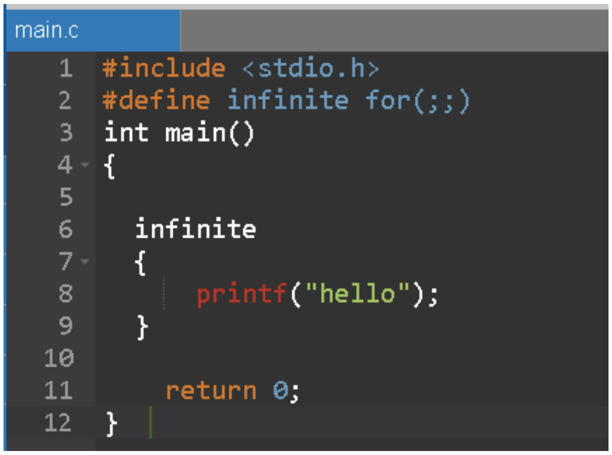
We can also use the goto statement to define the infinite loop.

```
infinite_loop;

// body statements.

goto infinite_loop;
```

In the above code, the goto statement transfers the control to the infinite loop.

5. Macros

We can also create the infinite loop with the help of a macro constant. Let's understand through an example.

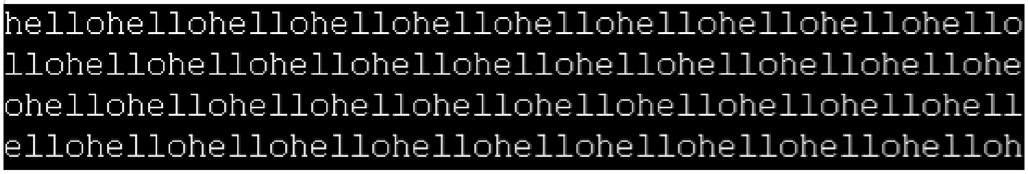


main.c

```
#include <stdio.h>
#define infinite for(;;)
int main()
{

  infinite
  {
      printf("hello");
  }

    return 0;
}
```

In the above code, we have defined a macro named as 'infinite', and its value is 'for(;;)'. Whenever the word 'infinite' comes in a program then it will be replaced with a 'for(;;)'.

Result/Output:



```
hellohellohellohellohellohellohellohellohellohello
llohellohellohellohellohellohellohellohellohellohe
ohellohellohellohellohellohellohellohellohellohell
ellohellohellohellohellohellohellohellohellohelloh
```

Till now, we have seen various ways to define an infinite loop. However, we need some approach to come out of the infinite loop. In order to come out of the infinite loop, we can use the break statement.

Let's understand through an example.

main.c

```c
#include <stdio.h>
int main()
{
 char ch;
 while(1)
 {
     ch=getchar();
     if(ch=='n')
     {
         break;
     }
     printf("hello");
 }
 return 0;
}
```

In the above code, we have defined the while loop, which will execute an infinite number of times until we press the key 'n'. We have added the 'if' statement inside the while loop. The 'if' statement contains the break keyword, and the break keyword brings control out of the loop.

Unintentional infinite loops

Sometimes the situation arises where unintentional infinite loops occur due to the bug in the code. If we are the beginners, then it becomes very difficult to trace them. Below are some measures to trace an unintentional infinite loop:

- We should examine the semicolons carefully. Sometimes we put the semicolon at the wrong place, which leads to the infinite loop.

main.c

```c
#include <stdio.h>
int main()
{
int i=1;
while(i<=10);
{
printf("%d", i);
i++;
}
return 0;
}
```

In the above code, we put the semicolon after the condition of the while loop which leads to the infinite loop. Due to this semicolon, the internal body of the while loop will not execute.

- We should check the logical conditions carefully. Sometimes by mistake, we place the assignment operator (=) instead of a relational operator (= =).

```
main.c
1  #include <stdio.h>
2  int main()
3  {
4  char ch='n';
5  while(ch='y')
6  {
7   printf("hello");
8  }
9  return 0;
10 }
```

In the above code, we use the assignment operator (ch='y') which leads to the execution of loop infinite number of times.

- We use the wrong loop condition which causes the loop to be executed indefinitely.

```
main.c
1  #include <stdio.h>
2  int main()
3  {
4    for(int i=1;i>=1;i++)
5    {
6        printf("hello");
7    }
8  return 0;
9  }
```

The above code will execute the 'for loop' infinite number of times. As we put the condition (i>=1), which will always be true for every condition, it means that "hello" will be printed infinitely.

- We should be careful when we are using the break keyword in the nested loop because it will terminate the execution of the nearest loop, not the entire loop.

```c
#include <stdio.h>
int main()
{
  while(1)
  {
      for(int i=1;i<=10;i++)
      {
          if(i%2==0)
          {
              break;
          }
      }
  }
    return 0;
}
```

In the above code, the while loop will be executed an infinite number of times as we use the break keyword in an inner loop. This break keyword will bring the control out of the inner loop, not from the outer loop.

- We should be very careful when we are using the floating-point value inside the loop as we cannot underestimate the floating-point errors.

```c
#include <stdio.h>
int main()
{
  float x = 3.0;
while (x != 4.0) {
  printf("x = %f\n", x);
  x += 0.1;
}
    return 0;
}
```

In the above code, the loop will run infinite times as the computer represents a floating-point value as a real value. The computer will represent the value of 4.0 as 3.999999 or 4.000001, so the condition (x !=4.0) will never be false. The solution to this problem is to write the condition as (k<=4.0).

III. C break

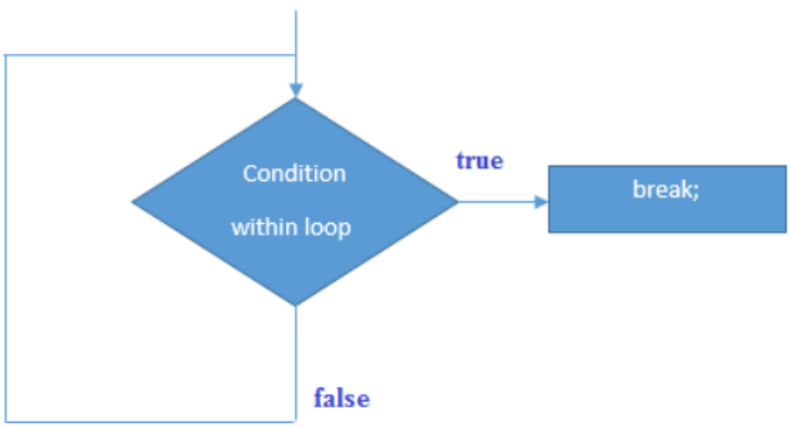
The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

```
//loop or switch case

break;
```

Flowchart of break in c:



LAB: See the following example:

```
#include<stdio.h>
#include<stdlib.h>
void main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
        break;
    }
    printf("came outside of loop i = %d",i);

}
```

Result/Output:

```
0 1 2 3 4 5 came outside of loop i = 5

...Program finished with exit code 0
Press ENTER to exit console.
```

C break statement with the nested loop

In such case, it breaks only the inner loop, but not outer loop.

```c
#include<stdio.h>
int main(){
int i=1,j=1;//initializing a local variable
for(i=1;i<=3;i++){
for(j=1;j<=3;j++){
printf("%d %d\n",i,j);
if(i==2 && j==2){
break;//will break loop of j only
}
}//end of for loop
return 0;
}
}
```

```
1 1
1 2
1 3
```

As you can see the output on the console, 2 3 is not printed because there is a break statement after printing i==2 and j==2. But 3 1, 3 2 and 3 3 are printed because the break statement is used to break the inner loop only.

break statement with while loop

Consider the following example to use break statement inside while loop.

```c
#include<stdio.h>
void main ()
{
    int i = 0;
    while(1)
    {
        printf("%d  ",i);
        i++;
        if(i == 10)
        break;
    }
    printf("came out of while loop");
}
```

```
0  1  2  3  4  5  6  7  8  9  came out of while loop
```

break statement with do-while loop

Consider the following example to use the break statement with a do-while loop.

```c
#include<stdio.h>
void main ()
{
   int n=2,i,choice;
   do
   {
       i=1;
       while(i<=10)
       {
           printf("%d X %d = %d\n",n,i,n*i);
           i++;
       }
       printf("do you want to continue with the table of %d , enter any non-zero value to continue.",n+1);
       scanf("%d",&choice);
    if(choice == 0)
       {
           break;
       }
       n++;
   }while(1);
}
```

Result/Output:

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20
do you want to continue with the table of 3 , enter any non-zero value to continue.0


...Program finished with exit code 0
Press ENTER to exit console.
```

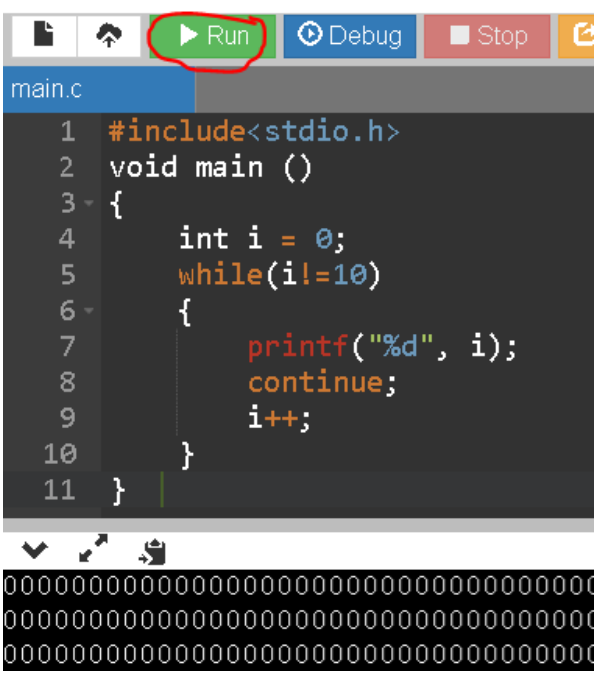
IV. C continue

The continue statement in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

Syntax:

```
//loop statements

continue;

//some lines of the code which is to be skipped
```

LAB: Continue statement example 1



LAB: Continue statement example 2

```
#include<stdio.h>
int main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5){//if value of i is equal to 5, it will continue the loop
continue;
}
printf("%d \n",i);
}//end of for loop
return 0;
}
```

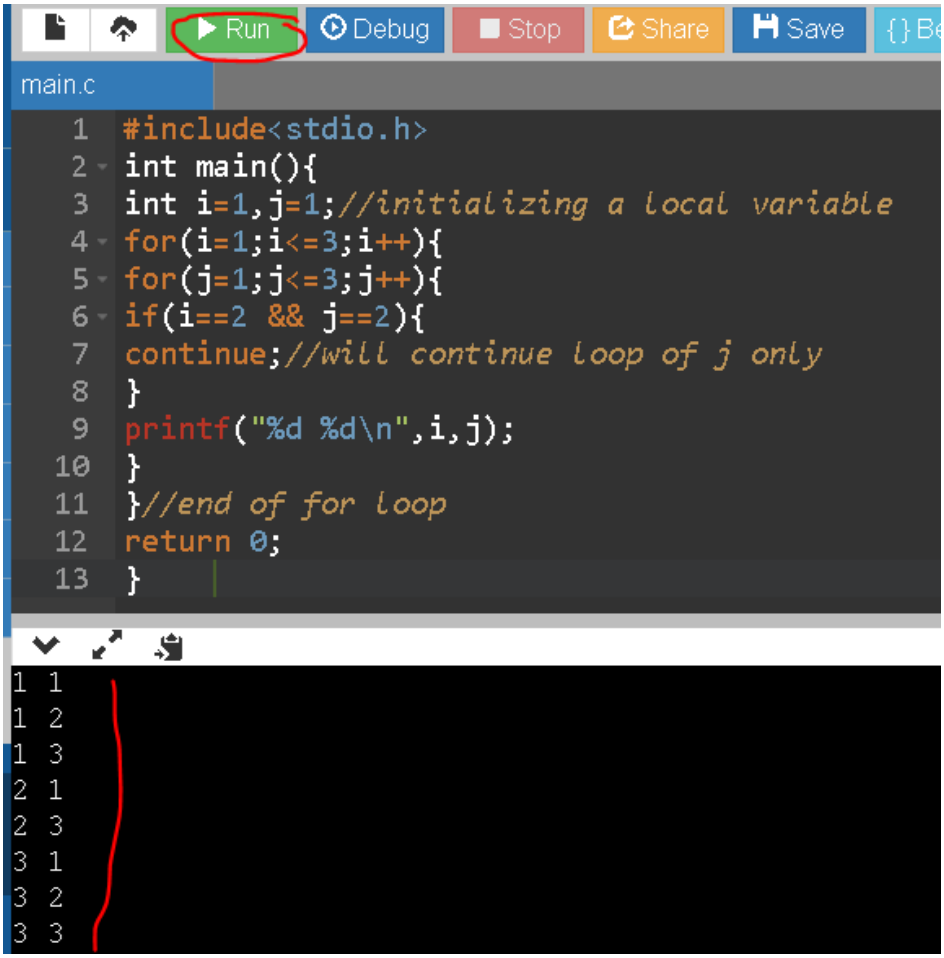
```
1
2
3
4
6
7
8
9
10

...Program finished with exit code 0
Press ENTER to exit console.
```

As you can see, 5 is not printed on the console because loop is continued at i==5

C continue statement with inner loop

In such case, C continue statement continues only inner loop, but not outer loop.

As you can see in the next example, 2 2 is not printed on the console because inner loop is continued at i==2 and j==2.



V. C goto

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complecated.

Syntax:

```
label:
//some part of the code;
goto label;
```

LAB: A simple example to use goto statement in C language:

```c
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
}
```

```
Enter the number whose table you want to print?3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30
```

When should we use goto?

The only condition in which using goto is preferable is when we need to break the multiple loops using a single statement at the same time. Consider the following example.

Run Debug Stop Share

main.c

```c
#include <stdio.h>
int main()
{
  int i, j, k;
  for(i=0;i<10;i++)
  {
    for(j=0;j<5;j++)
    {
      for(k=0;k<3;k++)
      {
        printf("%d %d %d\n",i,j,k);
        if(j == 3)
        {
          goto out;
        }
      }
    }
  }
  out:
  printf("came out of the loop");
}
```

```
0 0 0
0 0 1
0 0 2
0 1 0
0 1 1
0 1 2
0 2 0
0 2 1
0 2 2
0 3 0
came out of the loop
```

VI. Type Casting

Typecasting allows us to convert one data type into other. In C language, we use cast operator for typecasting which is denoted by (type).

Syntax:

```
(type)value;
```

Note: It is always recommended to convert the lower value to higher for avoiding data loss.

Without Type Casting:

```
int f= 9/4;

printf("f : %d\n", f );//Output: 2
```

With Type Casting:

```
float f=(float) 9/4;

printf("f : %f\n", f );//Output: 2.250000
```

LAB: A simple example to cast int value into the float.