CS2002D PROGRAM DESIGN
Lecture 4

Recall Control Structures

- **Control structures** control the flow of execution in a program or function.
- There are three kinds of execution flow:
  - **Sequence:**
  - **Selection:**
  - **Repetition:**
- Last class, we have discussed the repetitive structures: while , do while and few details on for

- We will today focus on for, nested loops, break, continue, event controlled loops
- We will also discuss Data Types

for loop….

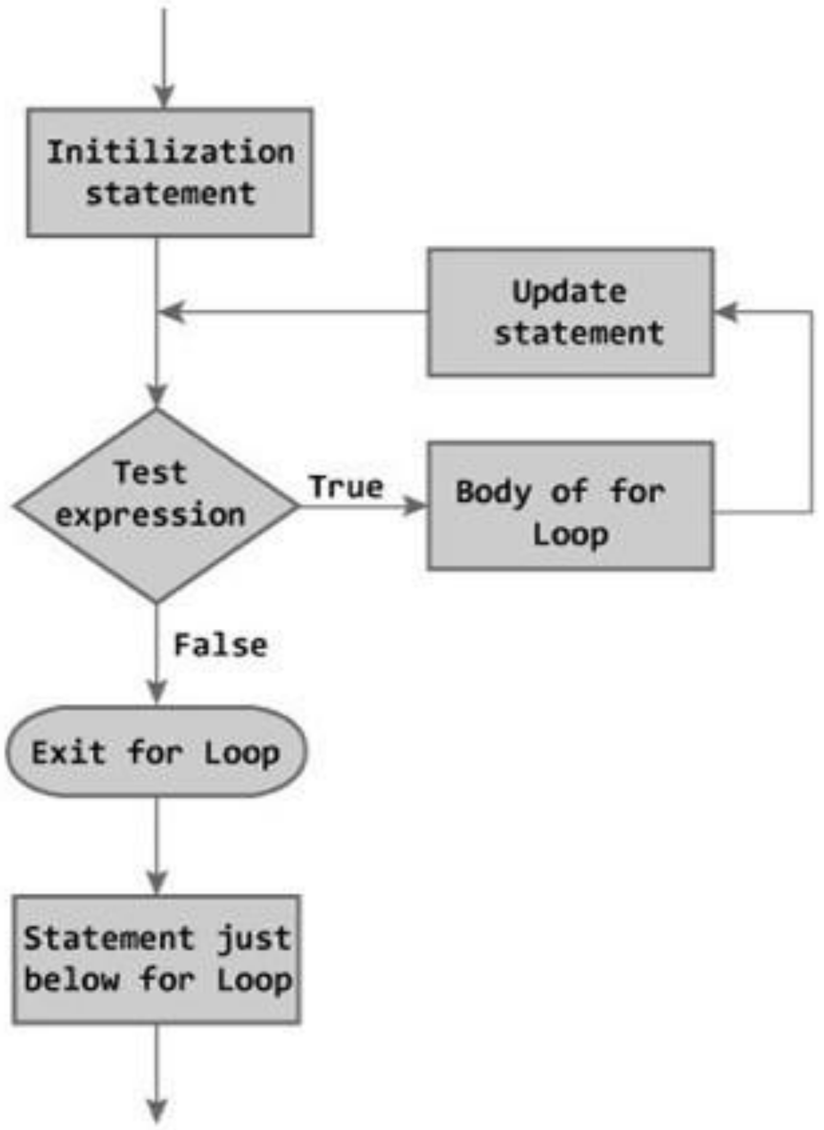


Figure: Flowchart of for Loop

Conversion

- ✓ *Anything that can be done with a "for" statement can also be done with an equivalent "while" statement.*

- ✓ **for (initialization; condition; update) statement;**

- The conversion is :

```
initialization;
while (condition)
{
        statement;
        update;
}
```

Comma operator to combine multiple expressions in for loop

```
for (i=0, j=10; i!=j; i++, j--)
{

   /*  statement block*/

 }
```

Expr1: *integer variables* ***i and j are initialized****, respectively, with 0 and 10 when the for statement is first encountered.*

Expr2: *relational expressions* ***i!=j*** *is evaluated and tested. If it evaluates to zero (false), the loop is terminated.*

Expr3: *After each iteration of the loop,* ***i is increased by 1*** *and* ***j is reduced by 1.***

Then the expression ***i!=j*** *is evaluated to determine whether or not to execute the loop again.*

Nesting of loops

- ✓ *Sometimes, within compound statements that are part of loops, there will be additional loops.*

- **Loops within loops are referred to as nested loops.**

- ✓ *There are outer loops (the loop encountered first) and inner loops (any loops embedded within an outer loop).*

Nested loops……While loop

```
initialize outer loop
while  (outer loop condition )
{       . . .

        initialize inner loop
        while (inner loop condition )
        {
                inner loop processing and update
        }
         . . .
}
```

Nested loops …. for loop

```
for (i=1; i<=3; i++) /* outer loop */
{
   printf("Start of  iteration %d of  the outer loop. \n", i);

        for (j=1; j<=4; j++) /* inner loop */
              printf(" Iteration %d of  inner loop.\n",j);

  printf("End of  iteration %d of  outer loop.\n", i);
}
```

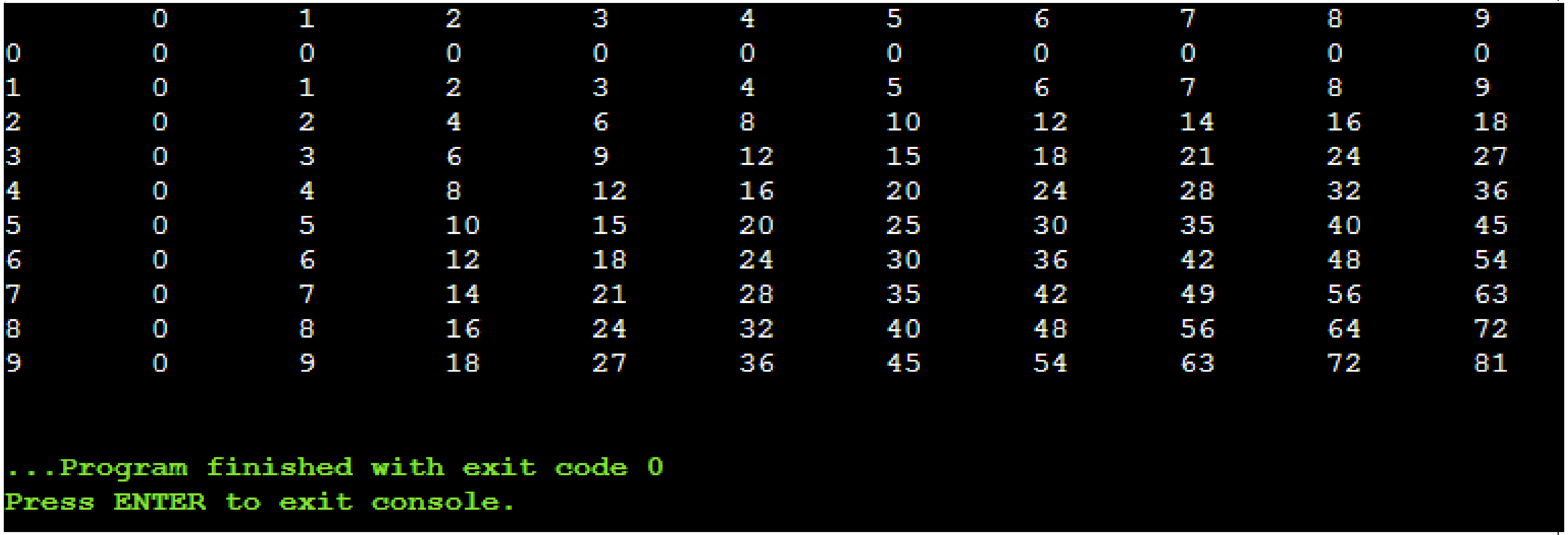
Example #1 :

```
int x, sum_digits, digit, temp;
 for (x = 1; x <= 1000; x++)
   {
        temp = x;
          sum_digits = 0;
        while (temp > 0)
          {
                 digit = temp % 10;
                 sum_digits = sum_digits + digit;
                 temp = temp / 10;
         }
        if (sum_digits == 5)
                 printf(“%d\n”, x);
   }
```

Example #2 : What does the code do?

```
int row, col;
printf("\t0\t1\t2\t3\t4\t5\t6\t7\t8\t9\n");
    for (row =  0; row <= 9; row++)
    {
            printf("%d", row);
            for (col =  0; col <= 9; col++)
            {
                    printf("\t%d", row*col);
            }
            printf("\n");
    }
```

Screenshot of Output



```
        0       1       2       3       4       5       6       7       8       9
0       0       0       0       0       0       0       0       0       0       0
1       0       1       2       3       4       5       6       7       8       9
2       0       2       4       6       8       10      12      14      16      18
3       0       3       6       9       12      15      18      21      24      27
4       0       4       8       12      16      20      24      28      32      36
5       0       5       10      15      20      25      30      35      40      45
6       0       6       12      18      24      30      36      42      48      54
7       0       7       14      21      28      35      42      49      56      63
8       0       8       16      24      32      40      48      56      64      72
9       0       9       18      27      36      45      54      63      72      81


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

- ✓ *Using the '\t' symbol within a string passed to "printf" causes a tab to be printed, and the computer skips to the start of the next 8 character column.*

- ✓ *The '\t' symbol is a special way of representing the tab character, in the same way that the '\n' symbol is a special way of representing the newline character.*

- ✓ *The first "printf" in this program skips the first column (since no characters appear to the left of the first tab), then prints out column headers 0 through 0 in the next 9 columns.*

- ✓ *We then loop through 9 rows of the table.*

- ✓ *At the beginning of each row, we print the row number.*

- ✓ *Then, we loop through 9 columns, and for each, we tab over to the column and print the product of the row number and column number.*

- ✓ *After the inner "for" loop (at the end of each iteration of the outer "for" loop), we print a '\n' which causes the program to start the next line.*

Trace the nested loop given

```
printf(“Max N! to print: “);
scanf(“%d”,&N);
for (I = 1; I <= N; I++) {
  fact = 1;
  for (J = 2; J <= I; J++)
    fact *= J;
  printf(“%d! = %d\n”,I,fact);
}
```

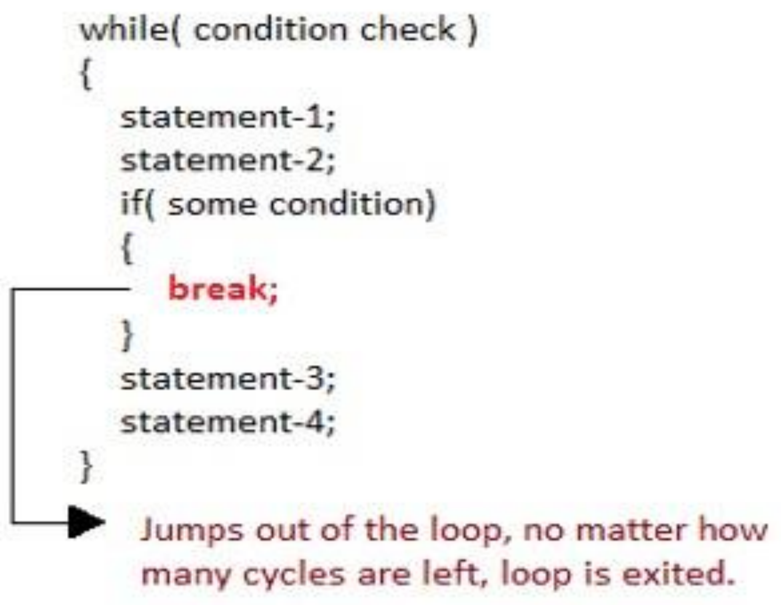
Tracing………..

Stmt	N	I	J	fact	output
1	4				
2		1			
3				1	
4			2		
6					1! = 1
2		2			
3				1	
4			2		
5				2	
4			3		
6					2! = 2
2		3			
3				1	
4			2		
5				2	
4			3		
5				6	
4			4		
6					3! = 6
2		4			
3				1	
4			2		
5				2	
4			3		
5				6	
4			4		
5				24	
4			5		
6					4! = 24
2		5			

Control of loop execution: Break Statement

- A loop construct, whether while, or do-while, or a for loop continues to **iteratively execute until the loop condition evaluates to false**
- There may be situations where it may be necessary **to exit from a loop even before the loop condition is reevaluated after an iteration**
- The break statement is used **to exit early** from all loop constructs (while, do-while, and for)

Example #1


```
while( condition check )
{
    statement-1;
    statement-2;
    if( some condition)
    {
       break;
    }
    statement-3;
    statement-4;
}
```
Jumps out of the loop, no matter how many cycles are left, loop is exited.

```
int x, y;
while (1)
{
        printf("Enter two numbers: ");
        scanf("%d %d", &x, &y);
        if (y ==  0) break;
                printf("%d /%d =  %d\n", x, y, x/y);
}
```

- ✓ *When you use "while(1)" (or any other non-zero constant), the loop should continue until some special statement inside the loop stops it.*
  - ✓ *Remember, when a non-zero constant is used as a boolean expression, it is interpreted as true.*

- ✓ *When the* **"break"** *statement is reached, the computer jumps to the first statement after the "while" loop, regardless of whether or not the condition (expression) being checked by the loop is true.*

- ✓ *The program goes into the loop no matter what and asks the user to enter the two numbers at the start of each iteration of the loop.*

- ✓ *When the user enters 0 as the second number, the loop is exited, and the program ends.*

Example #2

- ✓ *Break - terminates loop, execution continues with the first statement following the loop- for and while loops*

```
sum = 0;
for (k=1; k<=5; k++)
{
  scanf("%lf",&x);
    if (x > 10.0)
       break;
    sum +=x;
}
printf("Sum = %f
\n",sum);
```

```
sum = 0;
k=1;
while (k<=5)
{
    scanf("%lf",&x);
    if (x > 10.0)
       break;
    sum +=x;
    k++;
}
printf("Sum = %f \n",sum);
```

Break statement….

- *If you encounter a "break" statement in the middle of a nested loop, the control of the program jumps to the first statement after the innermost loop surrounding the "break" statement.*

✓ *For instance, let's say in the Example #2 of nested loops, you only want to print out half of the multiplication table, that below the diagonal line from the top left to the bottom right.*

✓ *There is no need to print the result of 2*7 if you are going to print the result of 7*2 anyway!*

Multiplication table program

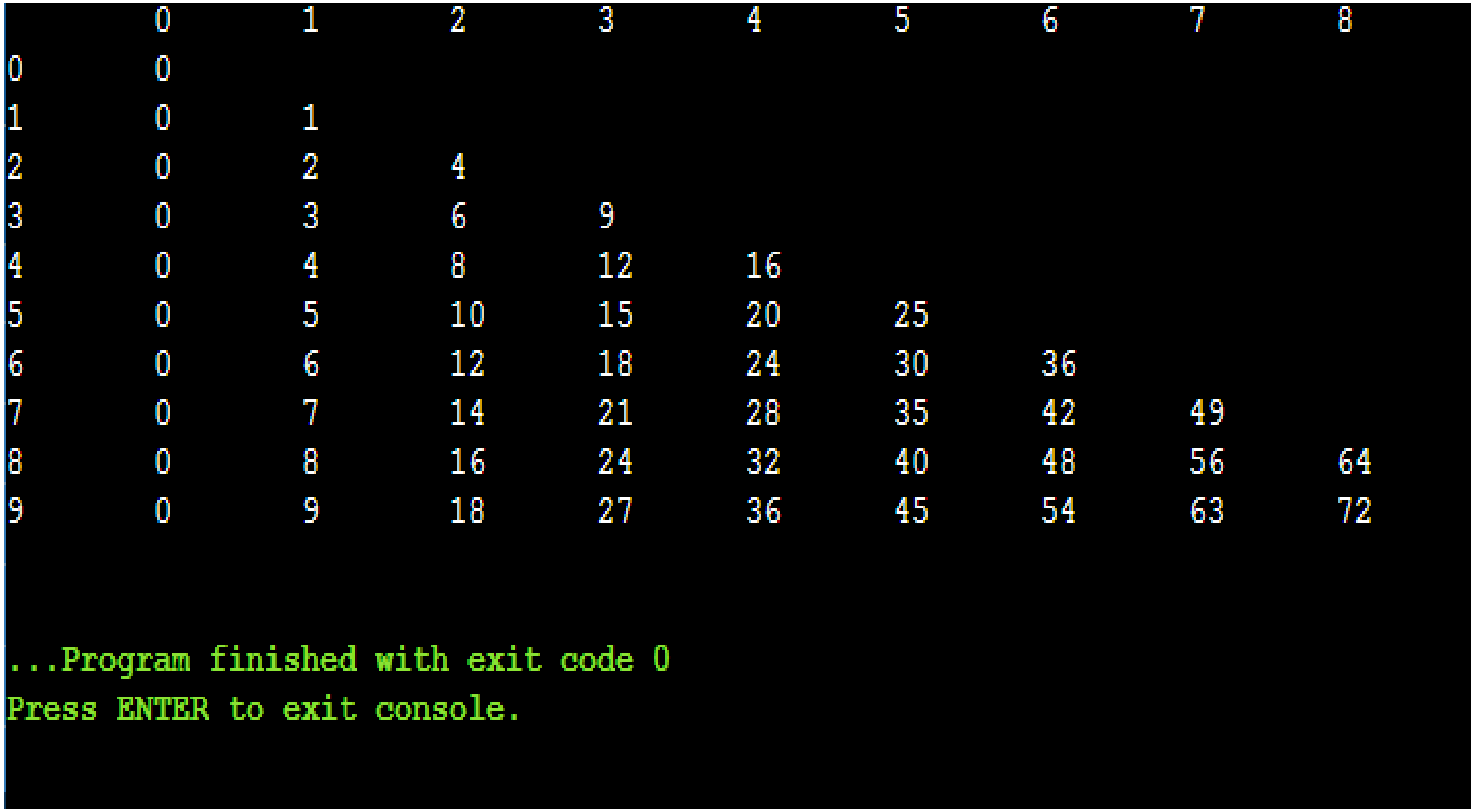
- ✓ *There is no need to print the result of 2*7 if you are going to print the result of 7*2 anyway!*
- ✓ *Can you can make this adjustment by adding one "if" statement to our multiplication program:?*

Multiplication table program revisited

```
int row, col;
 printf("\t0\t1\t2\t3\t4\t5\t6\t7\t8\t9\n");
    for (row =  0; row <= 9; row++)
    {
      printf("%d", row);
      for (col =  0; col <= 9; col++)
      {
           printf("\t%d", row*col);
           if (col ==  row)
                     break;
       }
        printf("\n");
    }
```

✓ *Now, for each row, once the column equals the row, we skip the rest of the inner "for" loop and jump to the line that prints the newline character. We then move on to the next row!*

Screenshot of the output



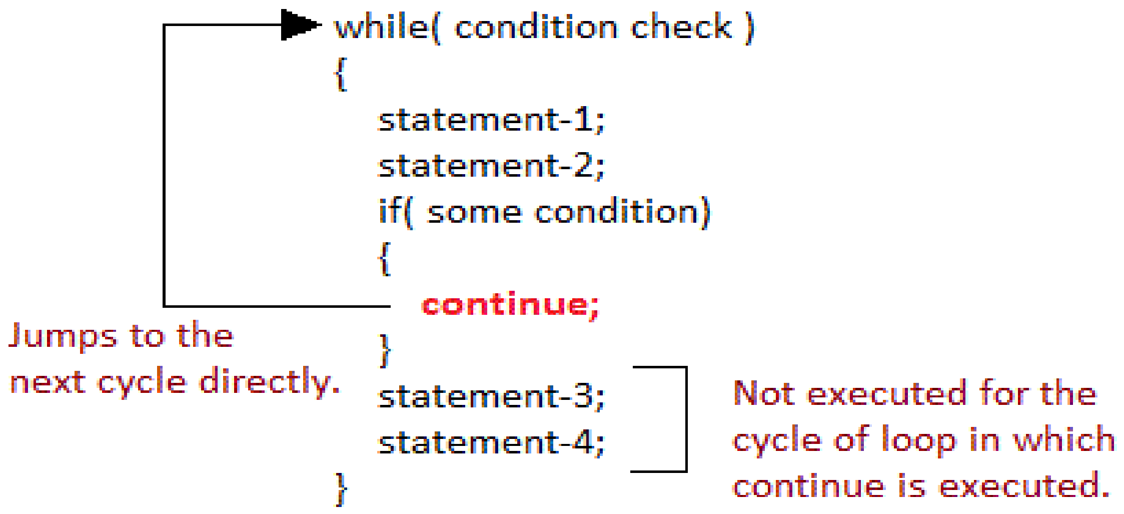
```
        0       1       2       3       4       5       6       7       8
0       0
1       0       1
2       0       2       4
3       0       3       6       9
4       0       4       8       12      16
5       0       5       10      15      20      25
6       0       6       12      18      24      30      36
7       0       7       14      21      28      35      42      49
8       0       8       16      24      32      40      48      56      64
9       0       9       18      27      36      45      54      63      72


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

Control of loop execution: Continue Statement

- The continue statement causes **all subsequent instructions in the loop body** (coming after the continue statement) **to be skipped**

- **Control passes back to the top of the loop** where the loop condition is evaluated again
  - ✓ *In case of a continue statement in a for loop construct, control passes to the reinitialization part of the loop, after which the loop condition is evaluated again.*



```
while( condition check )
{
    statement-1;
    statement-2;
    if( some condition)
    {
      continue;
    }
    statement-3;
    statement-4;
}
```

Jumps to the next cycle directly.
Not executed for the cycle of loop in which continue is executed.

Example #1

✓ **Continue** *forces next iteration of the loop, skipping any remaining statements in the loop- for and while loops*

```
sum = 0;
for (k=1; k<=5; k++)
{
   scanf("%lf",&x);
   if (x > 10.0)
      continue;
   sum +=x;
}
printf("Sum = %f \n",sum);
```

```
sum = 0;
k=1;
while (k<=5)
{
   scanf("%lf",&x);
   if (x > 10.0)
      continue;
   sum +=x;
   k++;
}
printf("Sum = %f \n",sum);
```

Example #2

```
int x;
 for (x =  1; x <= 20; x++)
 {
    if (x % 5 ==  0)
    continue;
    printf(“%d\n”, x);
 }
```

✓ *The first thing to note here is the test for divisibility by 5.*

✓ *Remember that the "%" is the modulus operator; it returns the remainder when the first operand is divided by the second operand.*
✓ ‘
✓ *If the remainder when "x" is divided by 5 is 0, then "x" is divisible by 5!*

✓ *When x is not divisible by 5, the condition of the “if” statement is false, so we reach the “printf” statement and print out the number.*

✓ *When "x" is 5, 10, or 15, the condition of the “if” statement is true, so we “continue”, or skip, to the end of the current iteration of the “for” loop, still do the update “x++”, and start the next iteration of the loop.*

✓ *When "x" is 20, we skip to the end of the current iteration of the “for” loop, still do the update “x++”, but now x will be 21, so the condition of the “for” loop is no longer met, and we end the loop.*

Example #3

```
int c;
printf(“Enter a character:\n(enter x to exit)\n”);
while {
 c = getch();
if (c == ‘x ’)
break;
 }
printf(“Break the infinite while loop. Bye!\n”);
```

```
Enter a character:
(enter x to exit)
H
I
x
Break the infinite while loop.
Bye!
```

Rewrite the code using continue statement

```
for (I =  0; I <  100;I++)
{
  if  (!((I% 2)== 1))
    printf("%d is even",I);
}
```

Rewrite the code using continue statement

```
for (I = 0; I < 100;I++)
{
 if (!((I % 2) == 1))
   printf("%d is even",I);
}
```

```
for (I = 0; I < 100; I++)
{
 if ((I % 2) == 1)
     continue;
 printf("%d is even",I);
}
```

Infinite loops

```
for ( ; ;)
{
        statement1;
        statement2;
          ..
          .
}
```

```
while {
          statement1;
          statement2;
            ..
            .
      }
```

Example #1

✓ Now consider the following program:

```
while (1)
{
        printf("Hello World!\n");
}
```

✓ *It prints "Hello Wold!" forever!*

✓ *This is called an infinite loop. Here, we created one on purpose, but normally, they are created by accident.*

- *What do you do when you are caught in an infinite loop?*
  - You press **Ctrl-C** on your keyboard.
  - **Pressing Ctrl-C will halt the execution of your C program!**

- *What happens if you use "while(0)" in your program?*
  - The loop is skipped no matter what.It is useless, but valid.

Here are five ways to exit a loop:

- ✓ ***The condition the loop depends on is not met at the time of the check.***
- ✓ ***A "break" statement is encountered.***
- ✓ ***Astatement that ends the current function, suchas "return", is encountered.***
- ✓ ***The program crashes.***
- ✓ ***The user presses Ctrl-C.***

Event controlled loops

✓ *read until input ends*
✓ *read until a number encountered*
✓ *search through data until item found*

- **Sentinel controlled** Keep processing data until a special value (which is not a possible data value) is entered to indicate that processing should stop

- **End-of-file controlled** Keep processing data as long as there is more data in the file
- **Flag controlled** Keep processing data until the value of a flag changes in the loop body

Example #1

✓ *Sentinel is negative blood pressure.*

```
int thisBP;
int total;
int count;
count = 1;
total = 0;

scanf( “%d”, &thisBP);

while (thisBP > 0)          // Test expression
{
  total = total + thisBP;
  count++;                  // Update
  }

printf(“The total = %d”, total);
```

Example #2

- ✓ The **value -999** is sometimes referred to as a ***sentinel* value**
- ✓ The value serves as the “guardian” for the termination of the loop
- ✓ Often a good idea to make the sentinel a constant

```
#define STOPNUMBER -999
while (number != STOPNUMBER)
 ...
```

Example#3

- ✓ **done** *is the flag, it controls the looping*

```
total = 0;
done = 0; /* done is set to 0 state */
do {
  scanf("%d",&num);
  if (num < 0) done = 1; /* done 1
  */
} while ((num != 0) && (!done));
```

Loop Testing and Debugging

- Test data should test all sections of the program
- Beware of infinite loops -- the program doesn't stop
- Check loop termination condition
- Use algorithm walk-through to verify that appropriate conditions occur in the right places
- Trace execution of loop by hand with code walk-through
- Use a debugger (if available) to run program in "slow motion" or use debug output statements

Data types in C

Data Types

- Data types in C is a system used for declaring variables or functions of different types.
- The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The data types in C can be classified as follows:

Sr.No.	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void** The type specifier *void* indicates that no value is available.
4	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Basic Types : Integer types

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	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

- To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator.
- The expressions *sizeof(type)* yields the storage size of the object or type in bytes

Basic Types : Floating point types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Enumerated types

- Enumeration is a user defined datatype in C language.
- It is used to assign names to the integral constants which makes a program easy to read and maintain.
- The keyword “enum” is used to declare an enumeration.

- Syntax of enum in C language:
- *enum enum_name{const1, const2, ... };*

Example of enumerated data types

```
/* Declaration */
enum sports
{
  cricket, football, hockey, tennis
};

/* Define two enums */
enum sports n1, n2;

/* Assign enum values */
n1 = cricket;
n2 = football;
```

The void Type

The void type specifies that no value is available. It is used in three kinds of situations:

Sr.No.	Types & Description
1	**Function returns as void** There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, **void exit (int status);**
2	**Function arguments as void** There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, **int rand(void);**
3	**Pointers to void** A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function **void *malloc(size_t size);** returns a pointer to void which can be casted to any data type.

Derived Types

- Array
- Structure
- Union
- Function
- Pointer

Declaration and initialization of a single dimensional array

```
#include<stdio.h>
int main(){
int i=0;
int marks[5]={20,30,40,50,60};//declaration and initialization of 
array.
//traversal of array.
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}
```

Declaration and initialization of a two dimensional array

```
#include<stdio.h>
int main(){
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
return 0;
}
```

Declaration and initialization of *Structure*

```
struct Point
{
   int x, y, z;
};

int main()
{
   // Examples of initialization using designated initialization
   struct Point p1 = {.y = 0, .z = 1, .x = 2};
   struct Point p2 = {.x = 20};

   printf ("x = %d, y = %d, z = %d\n", p1.x, p1.y, p1.z);
   printf ("x = %d", p2.x);
   return 0;
}
```

Brief introduction on pointers

- What is a pointer?
- Pointer is a variable that contains the address of a variable
- How do we declare a pointer variable?
- Syntax of pointer declaraion:

  data_type *pointer_name;
- Example: int *ptr;
- int a; //declaration of an integer

  int *ptr; //declaration of a pointer variable which points to an integer variable

  ptr = &a; //assigning the address of variable to the pointer variable—initialization of a pointer variable

Pointers

- * operator : indirection or dereferencing operator
- Every pointer points to a specific data type
- Does not point to an expression

```
void main() {
        int i=10;
       printf("\nValue of :%d“   ,i);
       printf("\nAddress of i :%d",&i);  }
```

Predict the output

```
main(){
        int x=10;  int  y=20;
        // variable ptr1 is declared as a pointer variable
        int  *ptr1=NULL; //it is initialized as  0
        printf( " First: %d %d %d", x,y, ptr1);
        //ptr1 is assigned x's address
        ptr1 = &x;
        // y is assigned the value ptr1 is pointing to
        y = *ptr1;
        printf( "Second: %d %d %d", x,y, *ptr1);
        // value at ptr1 is now 0
        *ptr1 =0;
        printf( "Third: %d %d %d", x,y, *ptr1);   }
```

Predict the output

```
main(){
        int x=10;  int  y=20;
        // variable ptr1 is declared as a pointer variable
        int  *ptr1=NULL; //it is initialized as  0
        printf( “ First: %d %d %d”, x,y, ptr1);
        //ptr1 is assigned x’s address
        ptr1 = &x;
        // y is assigned the value ptr1 is pointing to
        y = *ptr1;
        printf( “Second: %d %d %d”, x,y, *ptr1);
        // value at ptr1 is now 0
        *ptr1 =0;
        printf( “Third: %d %d %d”, x,y, *ptr1);   }
```

- **Ans: First: 10 20 0 Second:10 10 10 Third: 0 10 0**

Thank You