



CSN-101 (Introduction to Computer Science and Engineering)

Lecture 17: Basics of C Programming

Mr. Debraj Kundu and Ms. Sumit Sharma

Dr. Sudip Roy

Assistant Professor, Department of Computer Science and Engineering

Piazza Class Room: <https://piazza.com/iitr.ac.in/fall2019/csn101>

[Access Code: csn101@2019]

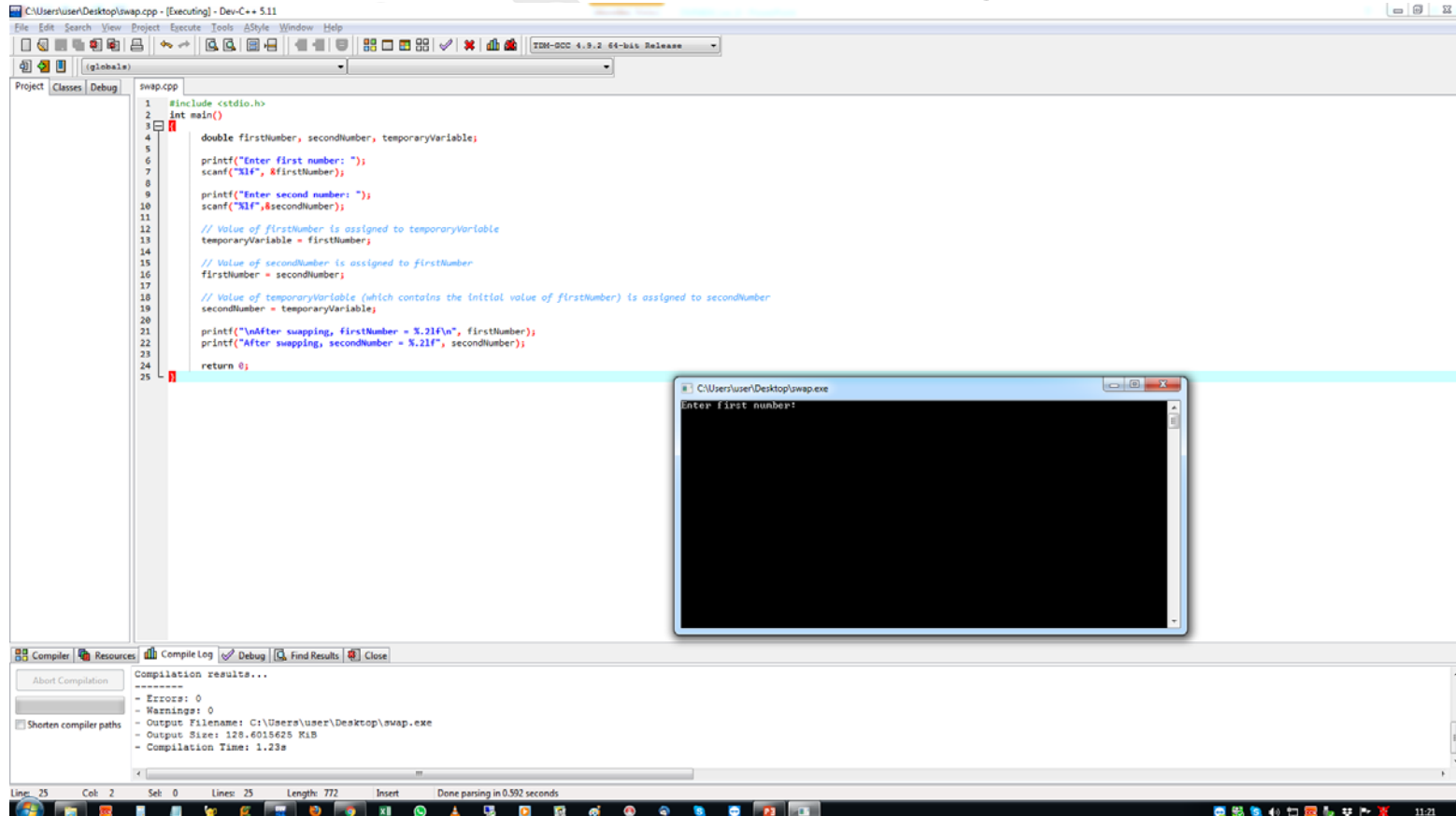
Moodle Submission Site: <https://moodle.iitr.ac.in/course/view.php?id=45>

[Enrollment Key: csn101@2019]



C Programming with Examples

- <https://www.programiz.com/c-programming/examples>
- Download Dev-C++ and install it in you PC
- Compile and Run some example C programs in Dev-C++



The screenshot displays the Dev-C++ IDE interface. The main editor window shows a C program named 'swap.cpp' with the following code:

```
1 #include <stdio.h>
2 int main()
3 {
4     double firstNumber, secondNumber, temporaryVariable;
5
6     printf("Enter first number: ");
7     scanf("%lf", &firstNumber);
8
9     printf("Enter second number: ");
10    scanf("%lf", &secondNumber);
11
12    // Value of firstNumber is assigned to temporaryVariable
13    temporaryVariable = firstNumber;
14
15    // Value of secondNumber is assigned to firstNumber
16    firstNumber = secondNumber;
17
18    // Value of temporaryVariable (which contains the initial value of firstNumber) is assigned to secondNumber
19    secondNumber = temporaryVariable;
20
21    printf("\nAfter swapping, firstNumber = %lf\n", firstNumber);
22    printf("\nAfter swapping, secondNumber = %lf\n", secondNumber);
23
24    return 0;
25 }
```

The 'Compiler' window at the bottom shows the compilation results:

```
Compilation results...
-----
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\user\Desktop\swap.exe
- Output Size: 128.6015625 KiB
- Compilation Time: 1.23s
```

An output window titled 'C:\Users\user\Desktop\swap.exe' is also visible, showing the prompt 'Enter first number:'.

Program Structure

- C program basically consists of the following parts:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

```
#include <stdio.h>

int main() {

    printf("Hello, World! \n");

    return 0;
}
```

- preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation
- *int main()* is the main function where the program execution begins
- */*...*/* will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program
- *printf(...)* causes the message "Hello, World!" to be displayed on the screen
- *return 0;* terminates the *main()* function and returns the value 0



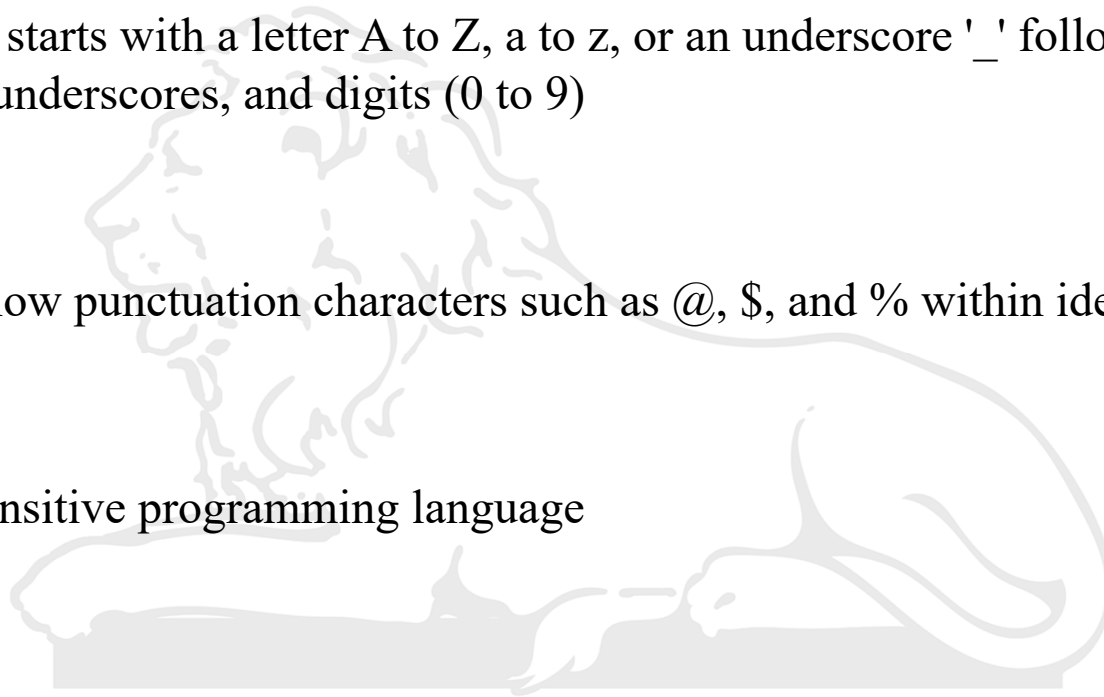
Compile and Execute

- Open a text editor and add the above-mentioned code
- Save the file as *hello.c*
- Open a command prompt and go to the directory where you have saved the fileType `gcc hello.c` and press enter to compile your code
- If there are no errors in your code, the command prompt will take you to the next line and would generate *a.out* executable file
- Now, type *a.out* to execute your program
- You will see the output "*Hello World*" printed on the screen

```
$ gcc hello.c  
$ ./a.out  
Hello, World!
```

Identifiers

- C identifier is a name used to identify a variable, function, or any other user-defined item
- An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9)
- C does not allow punctuation characters such as @, \$, and % within identifiers
- C is a case-sensitive programming language



Keywords

The following list shows the reserved words in C.

These reserved words may not be used as constants or variables or any other identifier names.

auto	else	long	switch
break	enum	register	typedef
case	extern	return	union
char	float	short	unsigned
const	for	signed	void
continue	goto	sizeof	volatile
default	if	static	while
do	int	struct	_Packed
double			

Data 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	-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.

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

Operators: Arithmetic

A = 10

B = 20

Operator	Description	Example
+	Adds two operands.	$A + B = 30$
-	Subtracts second operand from the first.	$A - B = -10$
*	Multiplies both operands.	$A * B = 200$
/	Divides numerator by denominator.	$B / A = 2$
%	Modulus Operator and remainder of after an integer division.	$B \% A = 0$
++	Increment operator increases the integer value by one.	$A++ = 11$
--	Decrement operator decreases the integer value by one.	$A-- = 9$



Operator: Relational

Operator	Description	Example
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is true.

Operators : Logical

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A && B) is true.

Operators : Bitwise and Assignment

Bitwise Operators

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A



Operators (cont.)

Misc Operators \mapsto sizeof & ternary

Operator	Description	Example
sizeof()	Returns the size of a variable.	sizeof(a), where a is integer, will return 4.
&	Returns the address of a variable.	&a; returns the actual address of the variable.
*	Pointer to a variable.	*a;
? :	Conditional Expression.	If Condition is true ? then value X : otherwise value Y



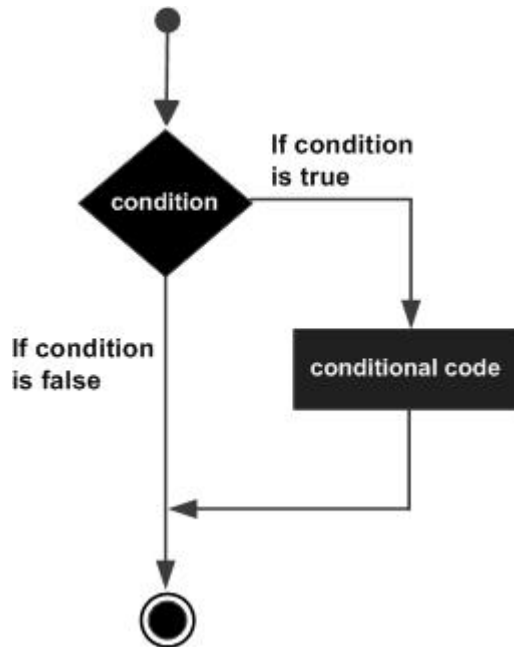
Operators (cont.)

Operators Precedence in C

Category	Operator	Associativity
Postfix	() [] -> . ++ --	Left to right
Unary	+ - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR		Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= =	Right to left
Comma	,	Left to right

Decision making

- If condition



```
if(boolean_expression) {  
    /* statement(s) will execute if the  
    boolean expression is true */  
}
```

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

```
int main () {
```

```
    /* local variable definition */
```

```
    int a = 10;
```

```
    /* check the boolean condition using if statement  
    */
```

```
    if( a < 20 ) {
```

```
        /* if condition is true then print the following */
```

```
        printf("a is less than 20\n" );
```

```
    }
```

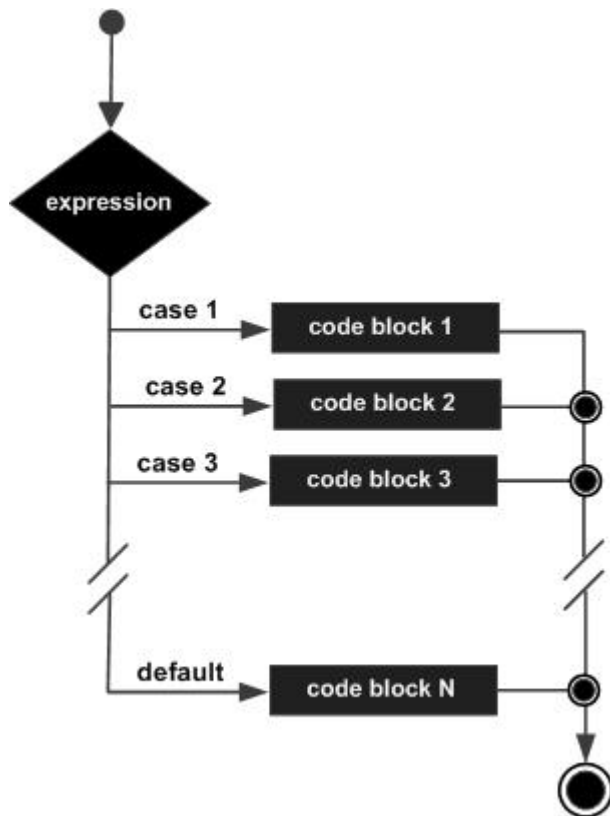
```
    printf("value of a is : %d\n", a);
```

```
    return 0;
```

```
}
```

Decision making (cont.)

- Switch statement



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

```
int main () {
    char grade = 'B';
```

```
    switch(grade) {
```

```
        case 'A' :
            printf("Excellent!\n" );
            break;
```

```
        case 'B' :
```

```
        case 'C' :
            printf("Well done\n" );
            break;
```

```
        case 'D' :
            printf("You passed\n" );
            break;
```

```
        case 'F' :
```

```
            printf("Better try
again\n" );
```

```
            break;
```

```
        default :
```

```
            printf("Invalid grade\n" );
        }
```

```
        printf("Your grade is %c\n",
grade );
```

```
        return 0;
```

```
    }
```

Output:

Well done

Your grade is B

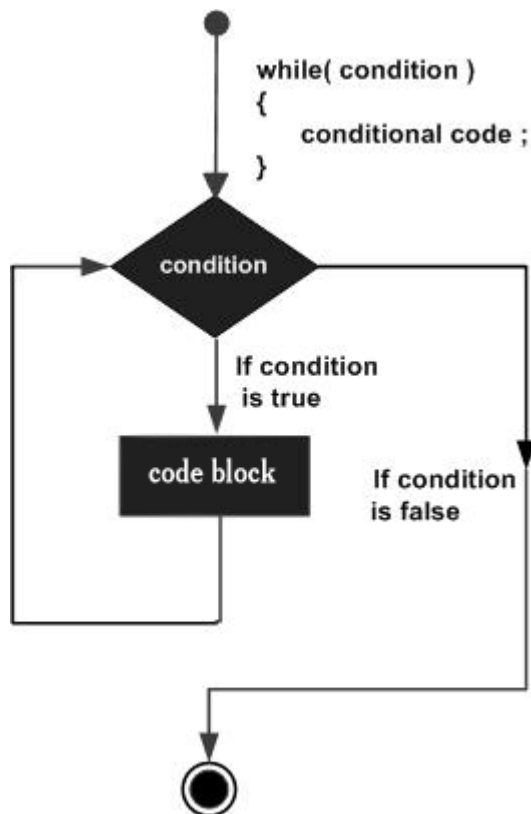
Ternary Operator

Exp1 ? Exp2 : Exp3;

- The value of a ? expression is determined like this –
- Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression.
- If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

```
int a = 10, b = 20, c;  
  
c = (a < b) ? a : b;  
  
printf("%d", c);
```


Loops: while



```
#include <stdio.h>

int main () {

    /* local variable definition */
    int a = 10;

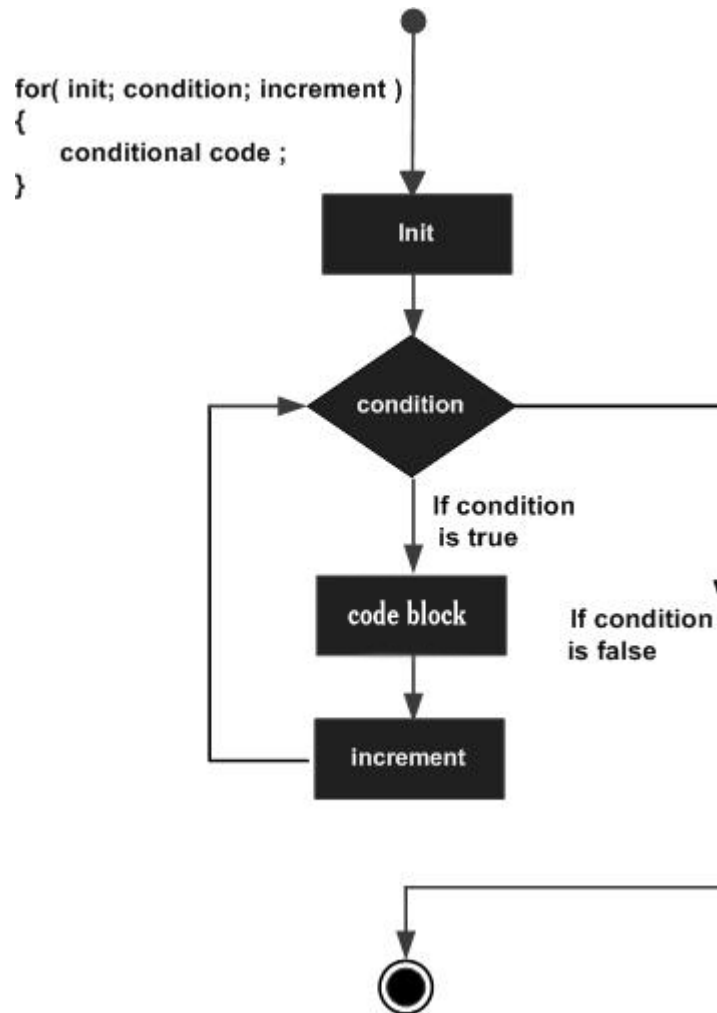
    /* while loop execution */
    while( a < 20 ) {
        printf("value of a: %d\n", a);
        a++;
    }

    return 0;
}
```

Output:

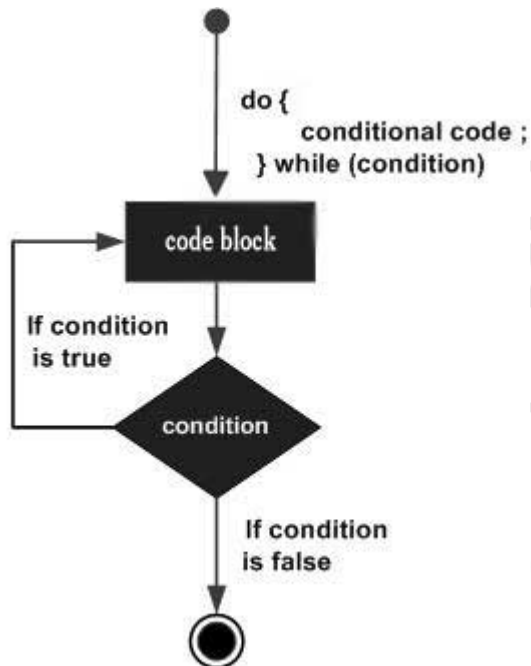
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

Loops : for



```
#include <stdio.h>  
  
int main () {  
  
    int a;  
  
    /* for loop execution */  
    for( a = 10; a < 20; a = a + 1 ){  
        printf("value of a: %d\n", a);  
    }  
  
    return 0;  
}
```

Loops : do while



```
#include <stdio.h>

int main () {

    /* local variable definition */
    int a = 10;

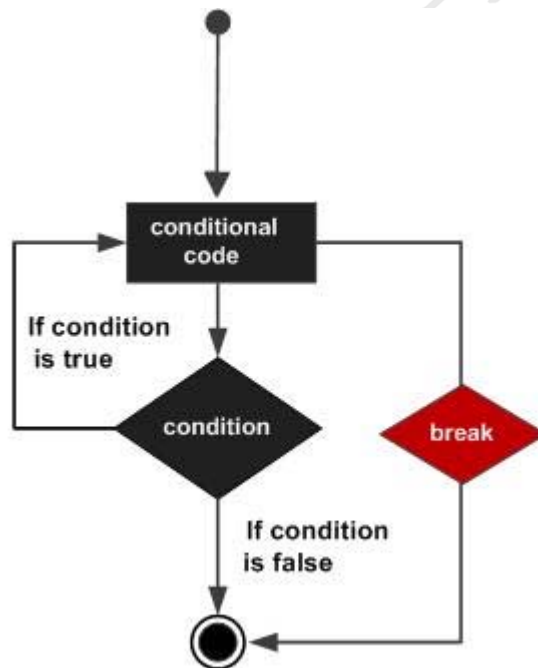
    /* do loop execution */
    do {
        printf("value of a: %d\n", a);
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

Loop Control Statements

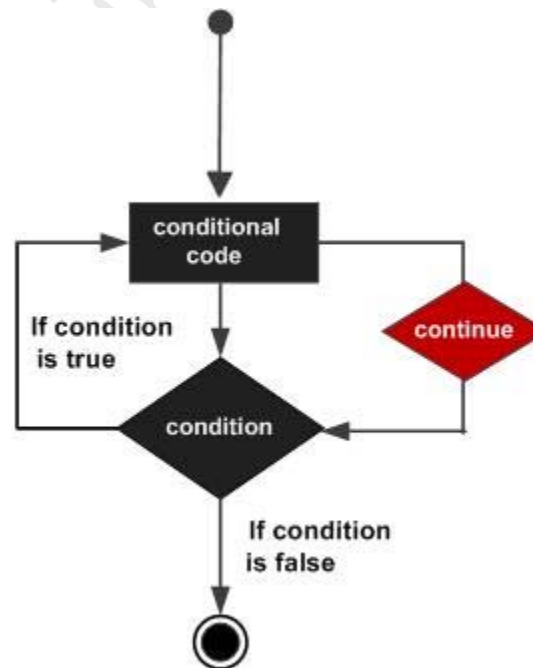
break

Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.



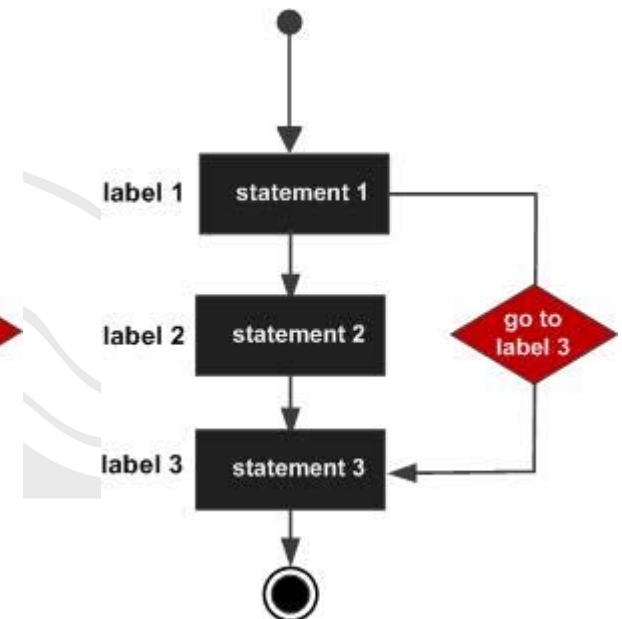
continue

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.



go to

Transfers control to the labeled statement.





Loop Control Statements (cont.)

```
#include <stdio.h>

int main () {

int a = 10;

while( a < 20 ) {

    printf("value of a: %d\n", a);
    a++;

    if( a > 15) {
        /* terminate the loop using break
        statement */
        break;
    }
}

return 0;
}
```

Output:

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15



Loop Control Statements (cont.)

```
#include <stdio.h>
int main () {
int a = 10;

/* do loop execution */
LOOP:do {

    if( a == 15) {
        /* skip the iteration */
        a = a + 1;
        goto LOOP;
    }
    printf("value of a: %d\n", a);
    a++;
}while( a < 20 );
return 0;
}
```

```
#include <stdio.h>
int main () {
int a = 10;

/* do loop execution */
do {

    if( a == 15) {
        /* skip the iteration */
        a = a + 1;
        continue;
    }
    printf("value of a: %d\n", a);
    a++;
}while( a < 20 );
return 0;
}
```

Output:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Infinite Loop

A loop becomes an infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose

```
#include <stdio.h>
int main () {
    for( ; ; ) {
        printf("This loop will run forever.\n");
    }
    return 0;
}
```

When the conditional expression is absent, it is assumed to be true. Initialization and increment expression can be present but C programmers more commonly use the for(; ;) construct to signify an infinite loop.

NOTE – You can terminate an infinite loop by pressing Ctrl + C keys.

Arrays

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type

Declaring Arrays

```
type arrayName [ arraySize ] ;
```

- *single-dimensional* array
- **arraySize** must be an integer constant greater than zero
- **type** can be any valid C data type

Initializing Arrays

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

	0	1	2	3	4
balance	1000.0	2.0	3.4	7.0	50.0

Arrays (cont.)

```
#include <stdio.h>

int main () {
    int n[ 10 ]; /* n is an array of 10 integers */
    int i,j;

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ ) {
        n[ i ] = i + 100; /* set element at location i
to i + 100 */
    }

    /* output each array element's value */
    for ( j = 0; j < 10; j++ ) {
        printf("Element[%d] = %d\n", j, n[j] );
    }
    return 0;
}
```

Output:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

Multi-dimensional Arrays

Declaring Arrays

type name [size1][size2]...[sizeN] ;

Initializing Arrays

```
int a[3][4] = {  
/* initializers for row indexed by 0 */  
    {0, 1, 2, 3} ,  
/* initializers for row indexed by 1 */  
    {4, 5, 6, 7} ,  
/* initializers for row indexed by 2*/  
    {8, 9, 10, 11}  
};
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

To be continued to next class...

