Basic C Tutorial

COMP 206 - Intro to Software Systems

History of C

- C is a general-purpose high level language that was originally developed by Dennis Ritchie
- C was invented to write an operating system called UNIX.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- C is a successor of B language which was introduced around 1970

Compilation Stages of a C Program

1. Preprocessing

All the preprocessor directives are executed by the program. They are lines that start with #.

#include <stdio.h>

This line loads the Standard Input Output Library

#define N 10 -> Macros

2. Compilation

Converts the C code into assembly language instructions.

It converts "abc.c" to a file called "abc.s"

3. Assembly

In this process a assembler is used to translate the assembly instructions to object code.

The output consists of actual instructions to be run by the target processor.

This step creates a file named abc.o, containing the object code of the program.

4. Linking

This step will get the external dependencies in the file and merge the missing pieces of code in the object code.

The result if this step is the final executable file a.out

Primitive Data Types in C

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	$-(2^{63})$ to $(2^{63})-1$	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	12		%Lf

Derived Data Types in C

- Arrays
- Pointers
- Structures
- Unions

We can use the **sizeof()** operator to check the size of a variable.

Operators in C

1. **Arithmetic Operators**

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

2. Assignment Operators

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

3. Relational Operators

Operator	Meaning of Operator	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

4. Logical Operators

Operator	Meaning of Operator	Example
&&	Logial 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.

5. Bitwise Operators

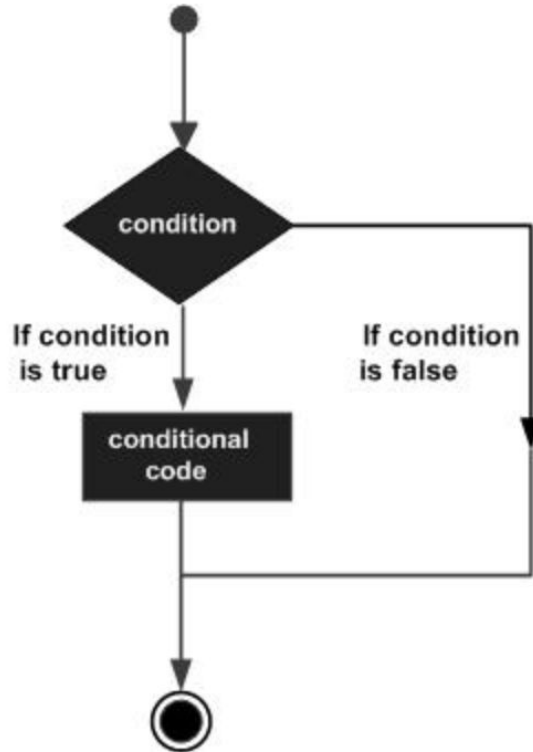| Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Other Operators

- Increment / Decrement Operator (++, --)
- Ternary Operator

  conditionalExpression ? expression1 : expression2

- sizeof() Operator

Decision Making
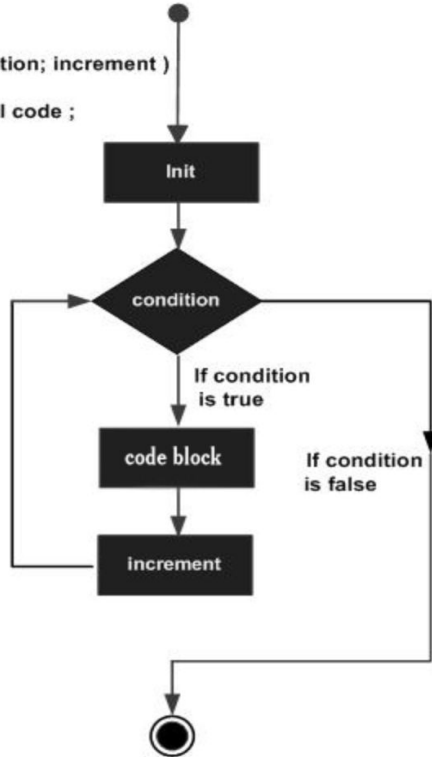

We use if - else statement

If (expr1) {

} else if (expr2) {

} else {
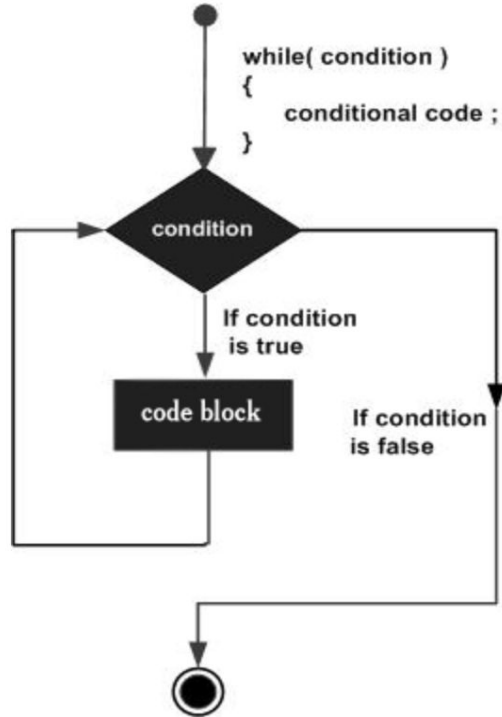
}

Loops

1. FOR Loops
2. WHILE loops
3. DO-WHILE loops

For Loop

```
for( init; condition; increment )
{
    conditional code ;
}
```
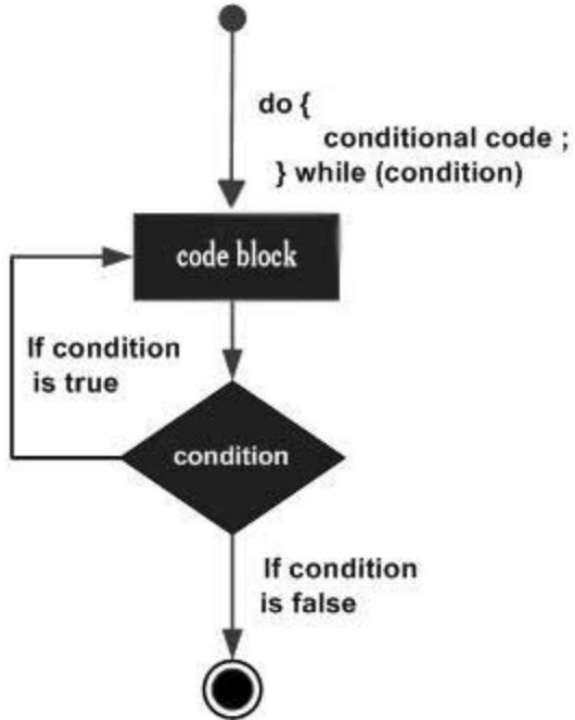

for (init; condition; increment) {

}

While Loop


```
while( condition )
{
    conditional code ;
}
```

While (condition) {

Conditional code

}

Entry Controlled Loop

Do-While Loop

Do {

Conditional

} while(condition);

Exit Controlled Loop

Nested Loops

```c
#include <stdio.h>
int main()
{
    int i=1,j;
    while (i <= 5)
    {
        j=1;
        while (j <= i )
        {
            printf("%d ",j);
            j++;
        }
        printf("\n");
        i++;
    }
    return 0;
}
```

What does it print ?

Loop Control Statements

1. break
2. continue
3. goto (not recommended to use this)

Note - Ensure we don't run into infinite loops

while(1) {

}

Functions

Reusability

Better management of code

```
return_type function_name ( parameter list ) {
    body of the function

    return return_type
}
```

Void functions have no return type

Call by Value

```
main() {
    int i = 10, j = 20;
    swapThemByVal(i, j);
    cout << i << " " << j << endl;    // displays 10  20
 }

 void swapThemByVal(int num1, int num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
 }
```

This creates a copy in memory of the actual parameter's value that is passed in, a copy of the contents of the actual parameter.

Call by Reference

```
main() {
    int i = 10, j = 20;
    swapThemByRef(i, j);
    cout << i << " " << j << endl;    // displays 20  10
}
void swapThemByRef(int& num1, int& num2) {
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

In pass by reference (also called pass by address), a copy of the address of the actual parameter is stored.

Arrays

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

```c
#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;
}
```

Multidimensional Arrays

```c
#include <stdio.h>

int main () {

   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ ) {

      for ( j = 0; j < 2; j++ ) {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }

   return 0;
}
```

Passing Array to Functions

```c
double getAverage(int arr[], int size) {

    int i;
    double avg;
    double sum = 0;

    for (i = 0; i < size; ++i) {
        sum += arr[i];
    }

    avg = sum / size;

    return avg;
}

int array[5] = {1000, 2, 3, 17, 50};
double avg;

/* pass pointer to the array as an argument */
avg = getAverage( array, 5 ) ;
```

Pointers

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location.

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Simple Example using Pointers

```c
#include <stdio.h>

int main () {

    int  var = 20;   /* actual variable declaration */
    int  *ip;        /* pointer variable declaration */

    ip = &var;  /* store address of var in pointer variable*/

    printf("Address of var variable: %x\n", &var  );

    /* address stored in pointer variable */
    printf("Address stored in ip variable: %x\n", ip );

    /* access the value using the pointer */
    printf("Value of *ip variable: %d\n", *ip );

    return 0;
}

Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

Passing Pointers to Functions in C

```c
#include <stdio.h>


/* function declaration */
double getAverage(int *arr, int size);

int main () {

    /* an int array with 5 elements */
    int array[5] = {1000, 2, 3, 17, 50};
    double avg;

    /* pass pointer to the array as an argument */
    avg = getAverage( array, 5 ) ;

    /* output the returned value  */
    printf("Average value is: %f\n", avg );
    return 0;
}
```

```c
double getAverage(int *arr, int size) {

    int i, sum = 0;
    double avg;
    for (i = 0; i < size; ++i) {
        Sum += arr[i];
    }

    avg = (double)sum / size;
    return avg;
}
```

Returning Pointers from Functions in C

```c
int * getRandom( ) {
    static int  r[10];
    int i;
    for ( i = 0; i < 10; ++i) {
        r[i] = rand();
        printf("%d\n", r[i] );
    }
    return r;
}


/* main function to call above defined function */
int main () {
    /* a pointer to an int */
    int *p;
    int i;
    p = getRandom();
    for ( i = 0; i < 10; i++ ) {
        printf("*(p + [%d]) : %d\n", i, *(p + i) );
    }
    return 0;
}
```

Strings

Strings are one-dimensional array of characters terminated by a null character '\0'. Thus a null-terminated string contains the characters that comprise the string followed by a null.

```c
#include <stdio.h>

int main () {

    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("Greeting message: %s\n", greeting );
    return 0;
}
```

String Inbuilt Functions

Sr.No.	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.

```c
#include <stdio.h>
#include <string.h>
int main () {
    char str1[12] = "Hello";
    char str2[12] = "World";
    char str3[12];
    int  len ;
    /* copy str1 into str3 */
    strcpy(str3, str1);
    printf("strcpy( str3, str1) :  %s\n", str3 );
    /* concatenates str1 and str2 */
    strcat( str1, str2);
    printf("strcat( str1, str2):   %s\n", str1 );
    /* total length of str1 after concatenation */
    len = strlen(str1);
    printf("strlen(str1) :  %d\n", len );
    return 0;
}

strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

Recursion

```c
/* Factorial of a number using Recursion */

#include <stdio.h>

unsigned long long int factorial(unsigned int i) {

   if(i <= 1) {
      return 1;
   }
   return i * factorial(i - 1);
}

int  main() {
   int i = 12;
   printf("Factorial of %d is %d\n", i, factorial(i));
   return 0;
}
```

Sources

The source code of the programs, tables, and figures have been taken from
https://www.tutorialspoint.com/cprogramming/