C++ and Unix

Dr.A.Mohamed Mustaq Ahmed

UNIT - I

Introduction to Programming in C: Basic Data Types –Variables– Constants – Operators. **Selection Statements**: if, Nested if, if-else-Ladder, switch. **Iteration Statements**: for Loop, while Loop, do-while Loop.

Arrays: Array Initialization - Single-Dimensional Arrays – Two-Dimensional Arrays. **Functions**: General Form – Function Arguments – return Statement – Recursion. **Structures**: General Format. **Pointers**: Pointer Variables – Pointer Expressions.

Introduction to C Programming

C is a general-purpose computer programming language developed in 1972 by Dennis Ritchie at the Bell Telephone Laboratories for use with the Unix operating system.

C is a structured programming language, which means that it allows you to develop programs using well-defined control structure.

Character set

Character set is a set of alphabets, letters and some special characters that are valid in C language.

Alphabets

Uppercase: A B C..... X Y Z

Lowercase: a b c..... x y z

C accepts both lowercase and uppercase alphabets as variables and functions.

Digits

0 1 2 3 4 5 6 7 8 9

Special Characters

Special Characters in C Programming- , < > . _ () { }[ ] #?” ’

White space Characters

Blank space, new line, horizontal tab, carriage return and form feed

C Tokens

C tokens are the basic buildings blocks in C language which are constructed together to write a C program.

Each and every smallest individual units in a C program are known as C tokens.

C tokens are of six types. They are,

Keywords (eg: int, while),

Identifiers (eg: main, total),

Constants (eg: 10, 20),

Strings (eg: “total”, “hello”),

Special symbols (eg: (), {}),

Operators (eg: +, /,-,*)

C- Language keywords

Keywords are predefined reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. For example:
int money;
Here, *int* is a keyword that indicates *'money'* is a variable of type integer.
As C is a case sensitive language, all keywords must be written in lowercase. Here is a list of all keywords allowed in ANSI C.

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

C Identifiers

Identifier refers to name given to entities such as variables, functions, structures etc.
Identifier must be unique. They are created to give unique name to a entity to identify it during the execution of the program. For example:

int money;
double accountBalance;

Here, *money* and *accountBalance* are identifiers.
Also remember, identifier names must be different from keywords. You cannot use *int* as an identifier because *int* is a keyword.

◇***Rules for writing an identifier***

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore.
3. There is no rule on length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler.

DATA TYPE

DATA TYPE	RANGE OF VALUES
char	-128 to 127
Int	-32768 to +32767
float	3.4 e-38 to 3.4 e+38
double	1.7 e-308 to 1.7 e+308

Integer Type

- Integers are whole numbers with a machine dependent range of values.
- A good programming language as to support the programmer by giving a control on a range of numbers and storage space.
- C has 3 classes of integer storage namely short int, int and long int. All of these data types have signed and unsigned forms.
- A short int requires half the space than normal integer values.
- Unsigned numbers are always positive and consume all the bits for the magnitude of the number.
- The long and unsigned integers are used to declare a longer range of values.

Floating Point Types

- Floating point number represents a real number with 6 digits precision.
- Floating point numbers are denoted by the keyword float.
- When the accuracy of the floating point number is insufficient, we can use the double to define the number.
- The double is same as float but with longer precision.
- To extend the precision further we can use long double which consumes 80 bits of memory space.

Void and Character Types

Void Type:

- Using void data type, we can specify the type of a function. It is a good practice to avoid functions that does not return any values to the calling function.

Character Type:

- A single character can be defined as a defined as a character type of data.
- Characters are usually stored in 8 bits of internal storage.
- The qualifier signed or unsigned can be explicitly applied to char.
- While unsigned characters have values between 0 and 255, signed characters have values from –128 to 127.

Constants/Literals

A constant is a value or an identifier whose value cannot be altered in a program. As mentioned, an identifier also can be defined as a constant.

const double PI = 3.14

Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.

Below are the different types of constants you can use in C.

1. Integer constants

- An integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:
- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:

Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

2. Floating - point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

- -2.0
- 0.0000234
- -0.22E-5

Note: E-5 = 10-5

3. Character constants

A character constant is a constant which uses single quotation around characters.
For example: 'a', 'l', 'm', 'F'

4. String constants

String constants are the constants which are enclosed in a pair of double-quote marks.

For example:

"good" //string constant

"" //null string constant

" " //string constant of six white space

"x" //string constant having single character.

"Earth is round\n" //prints string with newline

5. Enumeration constants

Keyword enum is used to define enumeration types. For example:

enum color {yellow, green, black, white};

Here, color is a variable and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively.

6. Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming.

For example: newline (enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: \n is used for newline. The backslash (\ ) causes "escape" from the normal way the characters are interpreted by the compiler.

Escape Sequences	
Escape Sequences	Character
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
****	Backslash

Variable

Variable is a name of memory location where we can store any data.

It can store only single data (Latest data) at a time. In C, a variable must be declared before it can be used.

Variables can be declared at the start of any block of code, but most are found at the start of each function.

A declaration begins with the type, followed by the name of one or more variables. For example,

DataType Name_of_Variable_Name;

int a,b,c;

Variable Names

- Every variable has a name and a value.
- The name identifies the variable, the value stores data.
- There is a limitation on what these names can be. Every variable name in C must start with a letter; the rest of the name can consist of letters, numbers and underscore characters.
- C recognizes upper and lower case characters as being different. Finally, you cannot use any of C's keywords like main, while, switch etc as variable names.

Examples of legal variable names include:

x	result	outfile	bestyet
x1	x2	out_file	best_yet
power	impetus	gamma	hi_score

It is conventional to avoid the use of capital letters in variable names. These are used for names of constants. Some old implementations of C only use the first 8 characters of a variable name.

Local Variables

Local variables are declared within the body of a function, and can only be used within that function only.

Syntax:

```
Void main( )
{
int a,b,c;
}
Void fun1()
{
int x,y,z;
}
```

Here a,b,c are the local variable of void main() function and it can’t be used within fun1() Function. And x, y and z are local variable of fun1().

Global Variables

A global variable declaration looks normal, but is located outside any of the program's functions. This is usually done at the beginning of the program file, but after preprocessor directives.

The variable is not declared again in the body of the functions which access it.

Syntax:

```
#include<stdio.h>
int a,b,c;
void main()
{
}
Void fun1()
{
}
```

Here a,b,c are global variable .and these variable cab be accessed (used) within

Exercises:

1. Write a C program to print your name, date of birth. and mobile number
2. Write a C program to print the following characters in a reverse way.
Test Characters: 'X', 'M', 'L'
Expected Output:
The reverse of XML is LMX
3) Write a C program to display multiple variables
4) Write a C program to convert specified days into years, weeks and days.
Note: Ignore leap year.
Test Data :
Number of days : 1329
Expected Output :
Years: 3
Weeks: 33
Days: 3
5) Write a C program to calculate the distance between the two points.
Test Data :
Input x1: 25 Input y1: 15
Input x2: 35 Input y2: 10
Expected Output:Distance between the said points: 11.1803

Operators in C

An operator is a symbol which operates on a value or a variable. For example: + is an operator to perform addition.

C programming has wide range of operators to perform various operations. For better understanding of operators, these operators can be classified as:

- Arithmetic Operators
- Increment and Decrement Operators
- Assignment Operators
- Relational Operators
- Logical Operators
- Conditional Operators
- Bitwise Operators
- Special Operators

Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

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

Arithmetic Operators

Example #1: Arithmetic Operators

```
#include <stdio.h>
int main()
{
    int a = 9,b = 4, c;
    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c=a/b;
    printf("a/b = %d \n",c);
    c=a%b;
    printf("Remainder when a divided by b = %d \n",c);

    return 0;
}
```

Ex2: Increment & Decrement Operators

```
// C Program to demonstrate the working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    return 0;
}
```

Assignment Operators

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

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

int c = 2, d = 3, e = 4, f = 6, g = 8

c+=2	c=?	f/=3	f=?
d-=1	d=?	g%=4	g=?
e*=2	e=?		

Ex 3: Assignment Operators

```
// C Program to demonstrate the working of assignment operators
#include <stdio.h>
int main()  {
    int a = 5, c;
    c = a;
    printf("c = %d \n", c);
    c += a; // c = c+a
    printf("c = %d \n", c);
    c -= a; // c = c-a}
```

Output

```
c = 5
c = 10
c = 5
```

Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

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

Ex: 4 Relational Operators

```
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10;
    printf("%d == %d = %d \n", a, b, a == b); // true
    printf("%d == %d = %d \n", a, c, a == c); // false
    printf("%d > %d = %d \n", a, b, a > b); //false
    return 0;
}
```

Output

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
```

Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

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.

Example #5: Logical Operators

```
#include <stdio.h>
void main() {
    int a = 5, b = 5, c = 10, result;
    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) equals to %d \n", result);
    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d \n", result);
    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) equals to %d \n", result); }
```

Output (a == b) && (c > b) equals to 1

(a == b) && (c < b) equals to 0

(a == b) || (c < b) equals to 1

Bitwise Operators

During computation, mathematical operations like: addition, subtraction, addition and division are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

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

Comma Operator

Comma operators are used to link related expressions together. For example:

int a, c = 5, d;

The sizeof operator

The sizeof is an unary operator which returns the size of data (constant, variables, array, structure etc).

Example #6: sizeof Operator

```
#include <stdio.h>
int main() {
    int a, e[10];
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));}
```

Output

Size of int = 4 bytes

Size of float = 4 bytes

Size of double = 8 bytes

Ternary Operator (?:)

A conditional operator is a ternary operator, that is, it works on 3 operands.

Conditional Operator Syntax

conditionalExpression ? expression1 : expression2

The conditional operator works as follows:

- The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
- If conditionalExpression is true, expression1 is evaluated.
- If conditionalExpression is false, expression2 is evaluated.

Example: y = (x ==1 ? 2 : 0) ;

```
#include <stdio.h>
int main() {int y,x;  x=2;
y = ( x ==1 ? 2 : 0 ) ;
printf("%d",y);     return 0;}
```

Expression in c

An expression is a combination of variables constants and operators written according to the syntax of C language. In C every expression evaluates to a value i.e., every expression results in some value of a certain type that can be assigned to a variable. Some examples of C expressions are shown in the table given below.

Example:

a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
3x2 +2x + 1	3*x*x+2*x+1
(x / y) + c	x / y + c

Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form

Variable = expression;

Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.

Example of evaluation statements are

x = a * b – c
y = b / c * a
z = a – b / c + d;

x=2+3*2 8,10,10,8 8

BODMAS (), (2/3*(2+1)*4)
* / (4/2+3*4)
+ - (2-3+4) (2+12)

Rules for evaluation of expression

- First parenthesized sub expression left to right are evaluated.
- If parenthesis are nested, the evaluation begins with the innermost sub expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub expressions.
- The associability rule is applied when two or more operators of the same precedence level appear in the sub expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When Parenthesis are used, the expressions within parenthesis assume highest priority.

C Library Functions

The C language is accompanied by a number of standard library functions which carry out various useful tasks. In particular, all input and output operations (e.g., writing to the terminal) and all math operations (e.g., evaluation of sines and cosines) are implemented by library functions.

C Header Files

<assert.h>	Program assertion functions
<ctype.h>	Character type functions
<locale.h>	Localization functions
<math.h>	Mathematics functions
<setjmp.h>	Jump functions
<signal.h>	Signal handling functions
<stdarg.h>	Variable arguments handling functions
<stdio.h>	Standard Input/Output functions
<stdlib.h>	Standard Utility functions
<string.h>	String handling functions
<time.h>	Date time functions

UNIT - II

Input/output functions – simple c programs –Flow of control – control structures – switch, break, continue, and go to statements – comma operator

Input/output functions

I/O FUNCTIONS and its DESCRIPTION

Scanf

The usually used input statement in C is scanf () function.

Getchar

It will accept a character from the console or from a file, displays immediately while typing and we need to press Enter key for proceeding.

Gets

It is used to scan a line of text from a standard input device. The gets() function will be terminated by a newline character.

Printf

The usually used output statement in C is printf () function.

Putchar

putchar function displays a single character on the screen.

Puts

It is used to display a string on a standard output device.

Scanf()

- This function is usually used as an input statement.
- The format string must be a text enclosed in double quotes. It contains the information for interpreting the entire data for connecting it into internal representation in memory.
  Example: integer (%d) , float (%f) , character (%c) or string (%s).
- The argument list contains a list of variables each preceded by the address list and separated by comma.
- The number of argument is not fixed; however corresponding to each argument there should be a format specifier. Inside the format string the number of argument should tally with the number of format specifier.
- **Syntax :**
  scanf(“format string”, argument list);

Getchar()

getchar() function will accept a character from the console or from a file, displays immediately while typing and we need to press Enter key for proceeding.

Syntax :

int getchar(void);

It returns the unsigned char that they read.If end-of-file or an error is encountered it return EOF.

This function reads a single character from the standard input device, the keyboard being assumed because that is the standard input device, and assigns it to the variable “a”.

Example:

a=getchar();

Gets()

- Gets function is used to scan a line of text from a standard input device. This function will be terminated by a newline character. The newline character won't be included as part of the string. The string may include white space characters.
- **Syntax :**

  char *gets(char *s);

- This function is declared in the header file stdio.h. It takes a single argument. The argument must be a data item representing a string. On successful completion, shall return a pointer to string s.
- Example:

  gets(str);

Printf

- The usually used output statement is printf (). It is one of the library functions.
- Syntax:

  printf (“format string”, argument list);

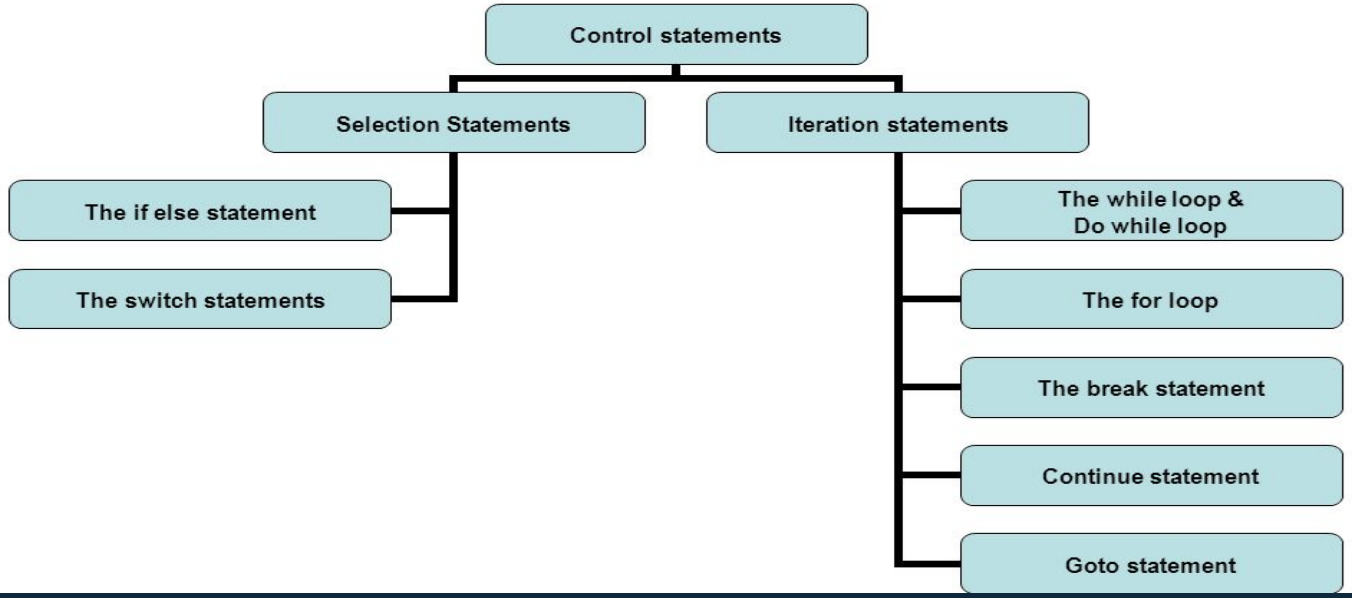
- Format string may be a collection of escape sequence or/and conversion specification or/and string constant. The format string directs this function to display the entire text enclosed within the double quotes without any change.

Flow of Control

Control Statements

Control statements in C are used to write powerful programs by;

1. Repeating important sections of the program.
2. Selecting between optional sections of a program.



If

It is the basic form where the if statement evaluate a test condition and direct program execution depending on the result of that evaluation.

Syntax:

```
If (Expression)
{
Statement 1;
Statement 2;
}
```

Where a statement may consist of a single statement, a compound statement or nothing as an empty statement.

The Expression also referred so as test condition must be enclosed in parenthesis, which cause the expression to be evaluated first, if it evaluate to true (a non zero value), then the statement associated with it will be executed otherwise ignored and the control will pass to the next statement.

If

Example:

```
if (a>b)
{
printf (“a is larger than b”);
}
```

IF- ELSE Statement

An if statement may also optionally contain a second statement, the ``else clause,'' which is to be executed if the condition is not met.

Syntax:

```
if (expression)
  {
    Block of statements;
  }
else
  {
    Block of statements;
  }
```

IF- ELSE Statement

example:

```
if(n > 0)
        average = sum / n;
else    {
        printf("can't compute average\n");
        average = 0;
        }
```

NESTED- IF Statement

It's also possible to nest one if statement inside another. (For that matter, it's in general possible to nest any kind of statement or control flow construct within another.) For example, here is a little piece of code which decides roughly which quadrant of the compass you're walking into, based on an x value which is positive if you're walking east, and a y value which is positive if you're walking north:

```
if(x > 0)       {
                if(y > 0)
                                printf("Northeast.\n");
                else            printf("Southeast.\n");
                }
else            {
                if(y > 0)
                                printf("Northwest.\n");
                else            printf("Southwest.\n");
                }
```

Switch Case

```
estimate(number)
int number;  /* Estimate a number as none, one, two, several, many */
{       switch(number) {
        case 0 :
                printf("None\n");
                break;
        case 1 :
                printf("One\n");
                break;
        case 2 :
                printf("Two\n");
                break;
        case 3 :
        case 4 :
        case 5 :
                printf("Several\n");
                break;
        default :
                printf("Many\n");
                break;        }
```

Each interesting case is listed with a corresponding action. The break statement prevents any further statements from being executed by leaving the switch. Since case 3 and case 4 have no following break, they continue on allowing the same action for several values of number.

Both if and switch constructs allow the programmer to make a selection from a number of possible actions.

Loops

Looping is a way by which we can execute any some set of statements more than one times continuously .In c there are mainly three types of loops are use :

- while Loop
- do while Loop
- For Loop

NESTED- IF Statement

It's also possible to nest one if statement inside another. (For that matter, it's in general possible to nest any kind of statement or control flow construct within another.) For example, here is a little piece of code which decides roughly which quadrant of the compass you're walking into, based on an x value which is positive if you're walking east, and a y value which is positive if you're walking north:

```
if(x > 0)       {
                if(y > 0)
                                printf("Northeast.\n");
                else            printf("Southeast.\n");
                }
else            {
                if(y > 0)
                                printf("Northwest.\n");
                else            printf("Southwest.\n");
                }
```

NESTED- IF Statement

It's also possible to nest one if statement inside another. (For that matter, it's in general possible to nest any kind of statement or control flow construct within another.) For example, here is a little piece of code which decides roughly which quadrant of the compass you're walking into, based on an x value which is positive if you're walking east, and a y value which is positive if you're walking north:

```
if(x > 0)       {
                if(y > 0)
                                printf("Northeast.\n");
                else            printf("Southeast.\n");
                }
else            {
                if(y > 0)
                                printf("Northwest.\n");
                else            printf("Southwest.\n");
                }
```

NESTED- IF Statement

It's also possible to nest one if statement inside another. (For that matter, it's in general possible to nest any kind of statement or control flow construct within another.) For example, here is a little piece of code which decides roughly which quadrant of the compass you're walking into, based on an x value which is positive if you're walking east, and a y value which is positive if you're walking north:

```
if(x > 0)       {
                if(y > 0)
                                printf("Northeast.\n");
                else            printf("Southeast.\n");
                }
else            {
                if(y > 0)
                                printf("Northwest.\n");
                else            printf("Southwest.\n");
                }
```

Thank You