

"C"

PROGRAMMING LANGUAGE

BY

FERAY ASAL

A PROJECT SUBMITTED TO THE POLYTECHNICAL SCHOOL COUNCIL

FOR

DIPLOMA IN COMPUTER PROGRAMMING

EASTERN MEDITERRANEAN UNIVERSITY

DEPARTMENT OF COMPUTER PROGRAMMING

AND

INFORMATION TECHNOLOGY

JANUARY 1992



TO MY DEAREST MUM & DAD

I am writing this letter to you as a part of my graduation project. I would like to thank the following persons for their help in finishing my graduation project: ARNOLD, JAMES, and JILL. I am writing this letter to you as a part of my graduation project. I would like to thank the following persons for their help in finishing my graduation project: ARNOLD, JAMES, and JILL. I am writing this letter to you as a part of my graduation project. I would like to thank the following persons for their help in finishing my graduation project: ARNOLD, JAMES, and JILL.

ACKNOWLEDGEMENT

I would like to thank to Mr. A. GUMUS for his kind help during the study of my graduation project.

I would like also to thank the following persons for their help in finishing my graduation project: ARMEN SARICA, Mr. ORHAN ERTUGRUL.

My special acknowlagment to Mr \ N . ICIL for assigning me to do the proper assignment to be my graduation project.

ABSTRACT

The programming language C is one of the most popular language in active use today. C is a practical language that offers the classical programming language concepts in new, simple and easy-to-use form.

This popularity has recently increased due to the proliferation of personal computers and the availability of powerful compiler and programming environments on them.

TABLE OF CONTENTS

	PAGE
Abstract	i
Table of Contents	ii
INTRODUCTION	1
CHAPTER 1. INTRODUCING C	2
1.1 What is C	2
1.2 Uses of C	4
1.3 C a structured language	5
CHAPTER 2. GENERAL OVERVIEW OF C	7
2.1 Function in C	7
2.2 The general form of C functions	8
2.3 The main() function	9
2.4 Function arguments	9
2.5 The printf() function	10

CHAPTER 3. C KEYWORDS	12
------------------------------------	-----------

3.1 C keywords list	12
---------------------------	----

CHAPTER 4. VARIABLES, CONSTANTS, OPERATORS,

AND EXPRESSION	13
-----------------------------	-----------

4.1 Variables	13
---------------------	----

4.2 Data types	13
----------------------	----

4.3 Declaration of variables	14
------------------------------------	----

4.4 Local variables	15
---------------------------	----

4.5 Formal Parameters	15
-----------------------------	----

4.6 Global Variables and extern	16
---------------------------------------	----

4.7 The static variable	17
-------------------------------	----

4.8 The register variables	18
----------------------------------	----

CHAPTER 5. ARRAY.....	19
------------------------------	-----------

5.1 Arrays in C	19
-----------------------	----

CHAPTER 6. ASSIGMENT STATEMENT	20
---	-----------

6.1 Assignment statement in C	20
-------------------------------------	----

CHAPTER 7. CONSTANTS	22
7.1 Constant in C	22
7.2 Backslash character constants	22
CHAPTER 8. OPERATORS	23
8.1 Operators in C	23
8.2 Arithmetic Operators	23
8.3 Relational and logical operators	24
8.4 Bitwise operators	24
8.5 The ? operators	25
8.6 The & and pointer operators	26
CHAPTER 9. EXPRESSIONS	27
9.1 Expression in C	27
9.2 Casts	27
CHAPTER 10. PROGRAM CONTROL STATEMENT	28
10.1 The if statement	28
10.2 The if-else-if ladder	29
10.3 The switch statement	30

CHAPTER 11. LOOP	32
11.1 Loop in C	32
11.2 The for loops	32
11.3 The while loop	32
11.4 The do-while loop	33
 CHAPTER 12. EXITING LOOPS USING BREAK	
AND EXIT	34
12.1 Introduction	34
12.2 The break statement	34
12.3 The exit() function	35
12.4 The continue statement	35
 CHAPTER 13. THE C CODE FOR A SIMPLE PROGRAM OF	
NOUGHTS & CROSSES	36
13.1 Coding in C	36
13.2 The headerfile "N" and "c.h"	36
13.3 The example program of C	37
Conclussion	41
Recomendation	42
References	43

INTRODUCTION

The purpose of this project is to give an introduction to C programming language.

This project assumes that you have some knowledge of programming. You should understand the general concepts of variables assignment statements, and loops.

CHAPTER 1

INTRODUCING C

WHAT IS C ?

C is often called middle-level computer language because it combines elements of a high-level language with the functionalism of assembler. Table 1-1 shows the levels of various computer languages including C.

A middle-level language gives programmers a minimal set of control and data manipulation statements that they can use to define high-level constructs. In contrast a high-level language is designed to try to give programmers everything they could possibly want already built into the language .

A low-level language forces programmers to define all program functions directly because nothing is built-in. Middle-level languages are sometimes thought of as building-block languages because the programmer first creates the

TABLE 1-1 Levels of Computer Languages

HIGH LEVEL	MIDDLE LEVEL	LOW LEVEL
Ada BASIC COBOL FORTRAN Pascal	C FORTH	Assembler

routines to perform all the programs necessary functions and then puts them together.

C allows the programmer to define routines to perform high-level commands. These routines are called functions.

As a middle-level language C manipulates the bits-bytes and addresses the computer functions with unlike a high-level language that can operate directly on strings of characters to perform a multitude of string functions. C can operate directly on characters. For example in BASIC, there are built-in statements to read and write disk files. In C, these procedures are performed by functions that are not part of the C language proper, but are provided in the C standard

library. These functions are special routines written in C that perform these operations.

C does have its benefits. It has very few statements to remember-only 28 keywords. This means that C compilers can be written reasonably easily since C operates on the same data types as the computer, the code output from a C compiler is efficient and fast. C can be used in place of assembler for most tasks.

C code is very portable. Portability means software written for one type of computer can be adapted to another type.

USES OF C

C was first used for system programming. System programming refers to a class of programs that either are part of or work closely with the operating system of the computer. System programs make the computer capable of performing useful work. These are examples of system programs that are often written in C:

- | | |
|-------------------|--------------------|
| -Operating system | -Language compiler |
| -Assemblers | -Text editors |
| -Print spoolers | - Network drivers |

-Modem programs

-Data bases

-Language interpreters

-Utilities

System programs often must run very quickly. Programs compilers can run almost as fast as those written in assembler. Since C code can be written more quickly than assembly code using C reduces costs tremendously.

Another reason that C is frequently used for system programming is that it is a programmer's language. Professional programmers seem to be attracted to C because it lacks restrictions and easily manipulates bits, bytes, and addresses. The system programmer needs C's direct control of the I/O and memory management functions.

C

A STRUCTURED

LANGUAGE

C is a structured language. The most distinguishing feature of a structured language is that it uses blocks. A block is a set of statements that, if successful will execute five discrete statements. If these statements can be grouped together and referenced easily, they form a block.

A structured language gives you a variety of programming possibilities. It supports the concept of subroutines with

local variables. A local variable is simply a variable that is known only to the subroutine in which it is defined. A structured language also supports several loop constructs, such as the while, do-while and for constructs. A structured language allows separately compiled subroutines to be used without being in the program proper. Structured languages tend to be more modern while the nonstructured are older. As you learn C programming languages, the difference between a structured and nonstructured language will become quite clear.

GENERAL OVERVIEW OF

C

Function in C :

The c language is based on the concept of building blocks. The building blocks are called functions. A C program is a collection of one or more functions. To write a program, you first create functions and then put them together.

TABLE 2-1 C version and BASIC version of program that prints HELLO

<pre>main() { printf("HELLO\n") }</pre>	<pre>10 PRINT "HELLO" 20 END</pre>
---	------------------------------------

A function is a subroutine that contains one or more C statements and performs one or more tasks. In well-written C

code each function performs only one task. Each function has

a name and a list of arguments that the function will receive. In general, you can give a function whatever name you please with the exception of main which is reserved for the function that begins program executions.

In the HELLO program of figure 2-1 both main () and printf () are functions.

The General Form Of C Functions:

The HELLO program introduces the general form of a C function. The program starts with main (). Then an opening brace signifies the beginning of the function followed by any statements that make up the function. In this program, the only statement is printf (). The closing brace signals the end of a function. Here it also marks the end of the program.

The general form of a function is

```
function-name (argument list)
argument-list declaration
{      opening brace begins the body of the
      function
      .
      . body of the function
      .
}      closing brace ends the function
```

As you can see, the first thing a function needs is the name. Inside the parentheses following the function name is

the list of arguments. Immediately following on the next line is the argument list declaration, which tells the compiler

what type of variable to expect. Next braces surround the body of the function. The body of the function is composed of the C statements that define what the function does.

The main () function:

The main() function is special because it is the first function called when your program executes a C program begins with a call to the main () function. The main () function can be anywhere in your program, although it is generally the first function for the sake of clarity. There can only be one main () in a program.

Function Arguments:

An argument is a value that is passed into a function. When a function is defined variables that will receive argument values must also be declared. These are called the formal parameters of the function. The return statement transmits the product back to the calling routine.

```
mul (x,y)                /*mul function*/
int x,y;                 /*here x and y are declared to
                          be integer variables*/
{
    return (x*y);        /*gives the product of the two
                          arguments*/
```

Each time mul() is called it will multiply the values of x and y. Table 1-3 presents a short program that uses the mul()

function. This program will print two numbers on the screen: 2 and 2340. The variables x,y,j-and k are not modified by the call to the mul () function. In fact,x and y in main () have no relationship to x and y in mul().

In C functions, arguments are always separated by commas.

Table 1-3 A PROGRAM USING THE MUL() FUNCTION

```
main
{
  int x,y,j,k;
  x=1;
  y=2;
  p=mul (x,y);
  printf ("% d",p); /*printf p in decimal*/
  j=234;
  k=10;
  p=mul (k,j)
  printf (" % d",p);
}
mul (x,y) /*mul function*/
int x,y; /*here x and y are declared to be
          integer variables*/ { return
                              (x*y);/*gives the product arguments*/
of the two arguments*/
}
```

The printf() function:

The general form of printf() is :

printf("control string" argument list)

In the printf() function, the control string contains format

commands that tell printf() how to display the remaining arguments there are, remember that each argument in the argument list is separated by a comma. The printf() allows a variety of format command as shown in table 1-4.

TABLE 1-4 A program using the mul() function

printf() code	format
%c	single character
%d	decimal
%e	scientific notation
%f	decimal floating point
%g	uses %e or %f, whichever is shorter
%o	octal
%s	string of character
%u	unsigned decimal
%x	hexadecimal

**C KEYWORDS LIST**

C has 28 keywords that may not be used as variable or function names. These words, when combined with the formal C syntax form the C programming language. The keywords are listed in Table 1-5 C requires that all keywords:

TABLE 1-5 Keyword list

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

lowercase. For example, Return will not be recognized as the keyword return.

**VARIABLES, CONSTANTS, OPERATORS,
AND EXPRESSIONS**

Variables:

Variable name in C can vary from one to several characters, with the first character being a letter and subsequent characters being either letters, numbers, or the underscore character. A variable may not be the same as a C keyword, and it should not have the same name as a function that you wrote or that is already in the C library.

Data Types:

There are seven built-in types of variables. The size and range of these data types vary with processor type with the implementation of the C compiler. The size and range information in table 1-6 will be correct.

TABLE 1-6 Variable Size and Range For Microcomputers

TYPE	BIT WIDTH	RANGE
char	8	0 to 255
int	16	-32768 to 32767
short int	8	-128 to 127
unsigned int	16	0 to 65535
long int	32	-4294967296 to 4294967295
float	32	approximately 6 digits of precision
double	64	approximately 12 digits of precision

Declaration of Variables:

All C variables must be declared before type are used. The syntax for declaring each type of variable is shown in the following examples:

```
int i;
short int si;
unsigned int ui;
long int li;
float f;
double d;
```

There are three basic places in a C program where variables will be declared: inside functions, in definition of function parameters, or outside of all functions. These variables are called local variables, formal parameters, and global variables.

Local Variables:

Local variables are declared inside a function. they may be referenced only by the statements that are inside the function in which the variables are declared. Local variables are not known to other functions outside their own; for example:

```
func1(( )
{
    int x;
    x=10;
}
func2()
{
    int x;
    x=-199;
}
```

Formal Parameters:

If a function has arguments must be declared. These are called the formal parameters of the function. The declaration occurs after the function name and before the opening brace; for example:

```
func1 ( first,last,ch)
int first,last;
char ch;
{
    int count;
    count= first*last;
    ch= 'a';
    .
    .
    .
}
```


In this example func1() has three arguments called first-last, and ch. You must tell C what type of variables these are by declaring them as shown in this fragment the formal parameters you declare are the same type as the mismatch, unexpected results can occur.

Global Variables And Extern:

Unlike local variables, global variables hold their value throughout the entire time your program is running. Global variables are created by declaring them outside of any function. They may be accessed by any expression in. Because C allows separately compiled modules of a large program to be linked together in order to speed up compilation, you must make sure that both files can reference the global variables. You can declare a global variable only once.

The extern modifier tells the compiler that the variables types and names that follow have already been declared elsewhere. In other words extern lets the compiler know what the types and names are for these global variables without actually creating them again. When the linker links the modules together, all reference to the external variables are resolved.

When you use a global variable inside a function tht is in the same file as the declaration for the global variable, you may select to use extern, although you don't have to. For example, this is the way to use this option:

```
int first, last;          /* global definition of first and
                           last*/

main()
{
    extern int first; /* optional use of
                       the extern declaration*/
}
```

The Static Variables:

The static variables are permanent variables within either their-own function or file. They differ from global variables because, while they are not known outside their function or file, they maintain their values between calles. This feature can make them very useful when you write generalized function and function libraries that are used by other programmers.

A good example of a function that might require such a variables is a number-series generator that produces a new number based on the last one.

```
series()
{
    static int series-num;
    series-num=series-num+23;
    return(series-num);
}
```


In this example, the variables series-num stays in existence between function calls instead of coming and going the way a normal local variable would. This means that each call to series() can produce a new member of the series based on the last number without declaring that variable globally.

The Register Variables:

C has last variable/declaration modifier that applies only to type int and char in most cases. The register modifier forces the C compiler to keep the value of variables declared with this modifier in the register of the CPU rather than in memory, where normal variables are stored. This means that operations on register variables can occur much faster than those on variables stored in memory because the value of register variables are held in the CPU and do not require a memory access. this makes register variables ideal for loop control. The register modifier can apply only to local variables and to the formal parameters in a function definition. Here is an example of how to declared register variables of int and char:

```
func1(s,u)
register int s;
register char u;
{
    float temp;
    register int counter;
    .
    .
    .
}
```


Array in C:

Arrays can be of any variable type. The general form of the array declaration is

```
type variable_name[number of elements];
```

For example, to make an array called `q` with ten integer elements, you would write :

```
int q[10];
```

Arrays elements are referenced by specifying the element number in brackets after name. To reference the first three elements of array `q`, you would write

```
q[0]  
q[1]  
q[2]
```

Unlike some BASICs, all arrays in C start with elements zero. This means that the ten elements of array `q` indexed from zero to nine.

ASSIGNMENT STATEMENTS

Assignment statement in C:

Up to this point, the examples have been assigning variables values without discussion. The general form of the assignment statement is as follow :

variable_name=expression;

where an expression may be as simple as a constant or as complex as an expression.

Type Conversion in Assignments:

Type conversion refers to the situation in which variables of one type are mixed with variables of another type. When this occurs in an assignment statement, the type conversion rule is very easy: the value of the right side of the assignment is converted to the type of the side, the target variable. For example :

```

int x;
char ch;
float f;
func()
{
    ch=x:
    x=f:
    f=ch:
    f=x:
}

```

TABLE 1-7 Assignment Type Conversion Rules

TARGET TYPE	EXPRESSION TYPE	POSSIBLE INFORMATION LOSS
char	short int	sign
char	int	high-order 8 bits
char	long int	high-order 24 bits
short int	int	high-order 8 bits
short int	long int	high-order 24 bits
int	long int	high-order 16 bits possibly more
float	double	precision; result rounded

CONSTANTS

Constant in C:

Constants in C refer to fixed values that may not be altered by a program. They can be of any type, as shown in table 1-8

TABLE 1-8 Example of Constants

DATA TYPE	CONSTANT EXAMPLES
char	'a' '\n' '9'
int	1 123 21000-234
long int	35000-34
short int	10-12 90
unsigned int	10000 987
float	123.23 4.34e-3
double	123.23 12312333-0.9876324

BACKSLASH CHARACTER CONSTANTS:

Enclosing all character constants in single quotes works for most printing characters, but a few such as the carriage return are impossible to enter from the keyboard. For this reason, C has created the special backslash character constants.

OPERATORS

Operators in C:

C is very rich in built_in operators. An operator is symbol that tells the compiler to perform specific mathematical or logical manipulations. C has three classes of operators: arithmetic, relational and logical, and bitwise. In addition, C has some special operators.

Arithmetic Operators:

Table 1-9 lists the arithmetic operators allowed in C. The operators +, -, *, and / all work the same way in C as they do in BASIC or any other computer language.

TABLE 1-9 Arithmetic Operators.

OPERATOR	MEANING
-	backspace
+	form feed
*	multiplication
/	division
%	modulo division
--	decrement
++	increment

Relational and Logical Operators:

In the terms relational operator and logical operator, relational refers to the relationship that values have with one another, and logical refers to the ways these relationship can be connected together. Table 1-10 shows the relational and logical operators.

TABLE 1-10 Relational and Logical Operators

RELATIONAL OPERATOR	ACTION
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal
==	equal
!=	not equal
LOGICAL OPERATOR	ACTION
&&	AND
	OR
!	NOT

Bitwise Operators:

Unlike many other languages, C support a complete set of bitwise operators. Since C is designed to take the place of assembly language for most programming tasks, C must support

all operations that can be done in assembler. Bitwise operations refers to the testing, setting, or shifting of the actual bits in an integer or character variable. These operations may not be used on type float or double. Table 1-11 lists these operators.

TABLE 1-11 The Bitwise Operators

OPERATOR	ACTION
&	AND
	OR
^	exclusive OR
~	one's complement
>>	shift right
<<	shift left

The ? Operator:

C allows a very powerful and convenient operator that can be used to replace statements of the if-then-else form. The ternary operator pair ?: takes the general form:

Exp1 ? Exp2 : Exp3

where Exp1, Exp2, and Exp3 are expressions

The ? operator works like this: Exp1 is evaluated. If it is true, Exp2 is evaluated and becomes the value of the expression. If Exp1 is false, Exp3 is evaluated and its value

becomes the value of the expression. For example, consider

```
x=10  
y= x>9 ? 100 : 200
```

The & and Pointer Operators:

In C, a pointer is the memory address of a variable. Pointers have two main functions in C: first, they can provide a very fast means of referencing array elements and, second, they allow C functions to modify their calling parameters. The first operator is &. It is a unary operator that returns the memory address of its operand.

```
m = &count;
```

The second operator is *. It is a unary operator that returns the value of the variable located at the address that follows. For example, if m contains the memory address of variable count, then

```
q = * m;
```

will place the value of count into q.

EXPRESSIONS

Expression in C:

Operators constants and variables are the constituents of expressions. An expression in C is valid combination of those pieces.

Casts:

It is possible to force an expression to be of a specific type by using a construct called a cast. The general form of cast is as follow :

(type) expression

where type is one of the standard C data types. For example, if you wished to make sur the expression $X/2$ was evaluated to type float, you could write it as follows:

(float) x/2

Casts are often considered operators. As an operator, a cast is unary and has the same precedence as any other unary operator.

PROGRAM CONTROL STATEMENT

The IF Statement:

The general form of the IF statement is as follow :

```
if(test condition)
    else statement2
```

where the objects of if and else are single statements. The else statement is optional. The objects of both if end else can be blocks of statements. The general form of the if with block of statements as objects is

```
if(test condition)
{
    statements/*block1*/
}
else
{
    statements/*block2*/
}
```

If the condition is true (that is, anything other than 0), statement1 or block1 will be executed; otherwise, if either exists, statement2 or block2 will be executed. Remember only one statement or block will be executed, not both.

For example, shows the C and BASIC versions of a program that prints the message ****Right**** when you guess the magic number. This program will be referred to as the "Magic Number Program" in later discussions.

This program uses the relational operator == to determine whether the guess entered matches the magic number. If it does, the message is printed on the screen.

```
main() /* magic number program */
{
    int magic = 123; /* magic number */
    int guess;
    guess = getnum(); /* read an integer from the
                       keyboard */
    if (guess==magic) printf("***Right***");
}
10      M=123
20      INPUT      G
30      IF  M=G  THEN  PRINT  "***Right**"
40      END
```

The if-else-if Ladder:

A common programming construct is the if-else-if ladder. It looks like this:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
.
else
    statement;
```

The conditions are evaluated from the top downward. As soon as a true condition is found, the statement associated with

it is executed, and the rest of the ladder is bypassed. If none of the conditions are true, the final else will be executed. The final else often acts as a default condition; that is, if all other condition tests fail, the last else statement is performed. If the final else is not present, no action will take place if all other conditions are false.

The switch Statement:

Although the IF-ELSE-IF ladder can perform a sequence of tests, it is hardly elegant. The code can be hard to follow and can confuse even the programmer at a later date. For reasons, C has a built-in multiple-branch decision statement called switch. The switch statement acts somewhat like the ON_GOTO and ON_GOSUB in BASIC, testing a variable successively against a list of integer or character constants. When a match is found, a statement or block of statements is executed. The general form of the switch statement is as follow :

```
switch(variable){
    case constant1:
        statement;
    case constant2:
        statement;
    case constant3:
        statement;
    .
    .
    .
    default:
        statement;
}
```

where default is performed if no matches are found. The default is optional, and if not present, no action takes place if all matches fail.

The switch differs from the if because switch can only test for equality, whereas the if can evaluate a relational or logical expression.

The switch statement is often used to process keyboard commands like the menu selection. As shown here with its BASIC equivalent, the function menu() will display a menu.

Here is a

switch statement that processes keyboard input. It uses a switch statement to process the input. The switch statement is used to process the input. The switch statement is used to process the input. The switch statement is used to process the input.

The last library

is the library. It is used to process the input. The library is used to process the input. The library is used to process the input.

main()

```
{
    int i;
    for(i=0; i<10; i++)
        printf("%d\n", i);
}
```

The while loop

Another form of a loop is the while loop. The while loop is used to process the input. The while loop is used to process the input. The while loop is used to process the input.

CHAPTER 11

LOOP

Loop in C:

In C and all other modern programming languages, loops allow a set of instructions to be performed until a certain condition is reached. This condition may be predefined, as in the for loop, or open-ended, as in the while and do-while loops.

The for Loops:

The for loop is used when you want to execute statements more than once.

```
main()
{
    int    x;
    for(x=1;x<=100;++*)
        printf ("HELLO  %d",x);
}
```

The while Loop:

Another form of a built-in loop is while. The general form of the statement is as follow :

```
while(condition)statement;
```

where statement may be a single statements or a block of statements that is to be repated. The condition may be any expresion,with true being any non-zero value. The statement is performed while the condition is true. When the condition becomes false program control passes to the line after the loop code.

```
wait-for-char()
{
    char    ch;
    ch=0    /* initialize ch */
    while (ch!='A')    ch=get char();
}
```

The do-while Loop:

Unlike the for and while loops that test the loop condition at the top of the loop, the do-while loop checks its condition at the bottom of the loop. The general form of the do-while loop is as follow:

```
do{
    statements;
} while (condition);
```

Although the braces are not necessary when only one statement is present, they are usually used to improve readability of do-while construct.

```
do {
    num=getnum()
} while (num>100);
```

EXITING LOOPS USING BREAK
AND EXIT()

Introduction:

The statement `break` and the library function `exit()` allow you to force an exit from inside a loop, bypassing the normal loop condition.

The break Statement:

When the `break` statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

```
main()
{
    int t;
    for(t=0;t++0 {
        printf ("%d",t);
        if(t==10) break;
    }
}
```

The exit() Function:

A second way to terminate a loop from inside is by using the `exit()` function, which is found in the standard library. Because the `exit()` function will cause immediate termination of your program and return to the operating system, its use is somewhat limited. The `exit()` function is traditionally called with an argument of 0 to indicate that termination is normal. Other arguments are used to indicate some sort of error. However, many micro-computer-based C compilers do not use the argument to `exit()` in any way, so often you will see `exit()` called with `)` as an argument under all circumstances.

```
main()
{
    if(!color-card()) exit(1)
    .
    .
    .
}
```

The continue Statement:

The `continue` statement works in almost the opposite way of the `break` statement: it forces the next iteration of the loop to take place, skipping any code in between.

```
do
{
    x=getnum();
    if(x,0) continue;
    printf ("%d ", x);
} while(x!=100);
```

THE C CODE FOR A SIMPLE
PROGRAM OF
NOUGHTS & CROSSES

Coding in C:

The code comes in 2 parts, a header file and a (definition file; N and C.h and N C.c respectively.

Note that the (library function clrscr() is provided by turbo C, but is not a standart ANSI C facilliting.

The headerfile "N" and "C.h":

```
# include < stdio.h .>

typedef enum {0,x,-} location type;

location type boardlocations [2] [2];

/* The header file inclues the standard input output
   function from the C libraries, defines an enumerated
   type, and constructs a 2d array variable of that
   type.*/
```



```
include <nandc.h> /* Include the headerfile */
* This function displays game instructions */
```

```
void display Instructions()
```

```
clrscr(); /* This is not standard ANSI C */
printf("A simple nought & crosses game \n");
printf("      by FERAY ASAL \n");
printf("\n");
printf("The game will allow two players to play noughts \n");
printf("& crosses and will inform the winning player of his victory \n");
printf("The board is displayed initially as a group of null spaces \n");
printf("i.e. '_' the list of possible positions is also given. \n");
printf("to make a move you simple enter a position \n");
printf("      Happy Playing !! \n");
```

```
/* end of function display instructions */
```

```
void displayboard() /* This function displays */
/* The state of the board */
```

```
int i = 0;
int j = 0;
printf("The possible board locations are: \n\n");
printf("1 2 3 \n 4 5 6 \n 7 8 9 \n");
printf("\n The state of the board is : \n\n");
for (i=0;i<3;i++)
{
    for (j=0;j<3;j++)
    {
        if (boardlocation [j] [i] = X)
            printf("X");

        else if (boardlocation [j] [i] = O)
            printf("O");
        else printf("_");

        if (j==2)
            printf("\n");
    };
};
```

```
/* end of function displayboard */
```

```
This function checks to see if someone has won */
```

```
int CheckForVictory (int player)
```

```
int i,j;
int null-result = 0;
locationType playermark = - ;
```

```
If (player==1)
    playermark=X;
```

```
If (player==2)
    playermark=O;
```

```

for (i=0;i<3;i++)
{
    j=0;
    if ((boardlocation [i] [j] = playermark)
        && (boardlocation [i] [j+1] = playermark)
        && (boardlocation [i] [j+2] = playermark))
        return player;
}

for (j=0;j<3;j++)
{
    i=0;
    if ((boardlocation [i] [j] = playermark)
        && (boardlocation [i+1] [j] = playermark)
        && (boardlocation [i+2] [j] = playermark))
        return player;
}

if (((boardlocation [1] [1] = playermark)
    && (boardlocation [2] [2] = playermark)
    && (boardlocation [0] [0] = playermark))
    || ((boardlocation [1] [1] = playermark)
    && (boardlocation [0] [2] = playermark)
    && (boardlocation [2] [0] = playermark)))
    return player;
else return null_result;
/* end of function check for victory */
/* This functions sets up the board and places the players */
/* nough or cross on it. */

id setboard (int position, int player)

int x = 0;
int error condition = 0;
int halt = 0;
int y = 0;
int l,m;
for(l=0;l<3;l++)
{
    for (m=0;m<3;m++)
    {
        boardlocation [l] [m] = -;
    };
};

while (!=halt)
{
    x=position;
    error condition=0;
    if (position <4)
        ((y=0) && (x--));
    else if ((position >3 ) && (position < 7))
        ((y=1) && (x=x-4));
    else if ((position > 7) && (position < 10))
        ((y=2) && (x=x-7));
    else
    {
        error condition = 1;
        position=error routine (error condition):

```

```

    };

if (boardlocations [x] [y] == -)
switch(player)
{
    case 1:
        boardlocations [x] [y] = x;
        break;

    case 2:
        boardlocations [x] [y] = o;
        break;

    default:
        break;
};
else
{
    errorcondition = 2;
    position = errorroutine (errorcondition);
};

if (errorcondition == 0)
    halt = 1;

};

```

```

int errorroutine (int errorcondition)

```

```

{
    int position=0;

```

```

    if (errorcondition=1)
    {

```

```

        printf("\n\n Incorrect position. Give an other position. \n");
        scanf("%d",&position);
    }

```

```

    else
    {

```

```

        printf("\n\n That position is already taken. \n");
        printf("Choose another \n");
        scanf("%d",&position);
    };

```

```

    return position;
}

```

```

id control()

```

```

{
    int winner=0;

```

```

    int player=0;

```

```

    int halt=0;

```

```

    int position=0;

```

```

    display Instructions();

```

```

    display Board();

```

```

    While (!halt)
    {

```

```

        printf ("Enter move player 1\n");
    }
}

```



```

scanf ("%d," & position);
player=1;
Set-Board (position,player);
display Board
winner=Check For Victory(player);
if (winner==1)
{
    halt=1;
    printf ("player 1 wins!\n");
    continue;
};
printf ("Enter move player2\n");
scanf ("%d",& position);
player=2;
Set-Board(position,player);
display Board();
winner=Check For Victory(player);
if (winner==2)
};
};
main()
{
    control ();

```

CONCLUSION

C has advantages over other languages.

Because C is a compact, efficient, flexible and expressive language. It has few keywords, but right control structures, powerful operators and easily-combined data types. This means that is easy to learn the language, easy to erite a C compiler, and programs written in this language are short, but also sometimes difficult to follow. C has suplemmented the use of assembly language programming on many systems.

C has disadvantages too.

Programs written in C may become somewhat encriptive, because of its rich operator set which reduces the program readability. C is not very strongly program errors, such as trying to index an array out of its bounds; in some cases the compiler can reorder evaluation of sub-expressions in expressions or expressions in argument lists, which may result in unexpected side effects; multiple uses of some symbols can lead to some programming mistakes, such as mixing up the equality and the assignment operators.

RECOMMENDATION

C is so usefull and good language for system programming.

If you want to learn C language we need enough time to learn C language exactly. I hope in futher this language will be taught to the student with the updating language.

REFERENCES

- 1 "The C Programming Languages" by
D. M. Ritchie.
- 2 "Standart C",by
P. J. Plauger, Jim Brodie.