**C Fundamentals**

***A brief history of C***

C evolved from a language called B, written by Ken Thompson at Bell Labs in 1970. Ken used B to write one of the first implementations of UNIX. B in turn was a descendant of the language BCPL (developed at Cambridge (UK) in 1967), with most of its instructions removed.

So many instructions were removed in going from BCPL to B, that Dennis Ritchie of Bell Labs put some back in (in 1972), and called the language C. The famous book *The C Programming Language* was written by Kernighan and Ritchie in 1978, and was the definitive reference book on C for almost a decade.

The original C was still too limiting, and not standardized, and so in 1983 an ANSI committee was established to formalise the language definition. It has taken until now (ten years later) for the ANSI (American National Standard Institute) standard to become well accepted and almost universally supported by compilers

**Data Types**

There are a number of ‘built-in’ data types in C. These are listed below.

Data types Size (bits) Range

Char or Signed Char 8 -128 to 127

Unsigned Char 8 0 to 255

Int or Signed int 16 -32768 to 32767

Unsigned int 16 0 to 65535

Unsigned or

Signed short int 8 -128 to 127

Unsigned short int 8 0 to 255

Unsigned or

signed long int 32 -2147483648 to 2147483647

Unsigned long int 32 0 to 4294967295

Float 32 3.4 e-38 to 3.4 e+38

Double 64 1.7e-308 to 1.7e+308

Long Double 80 3.4 e-4932 to 3.4 e+4932

**Escape Sequences**

There are several character *escape* sequences which can be used in place of a character

constant or within a string. They are:

**Escape sequence Meaning**

\a alert (bell)

\b backspace

\f formfeed

\n newline

\r carriage return

\t tab

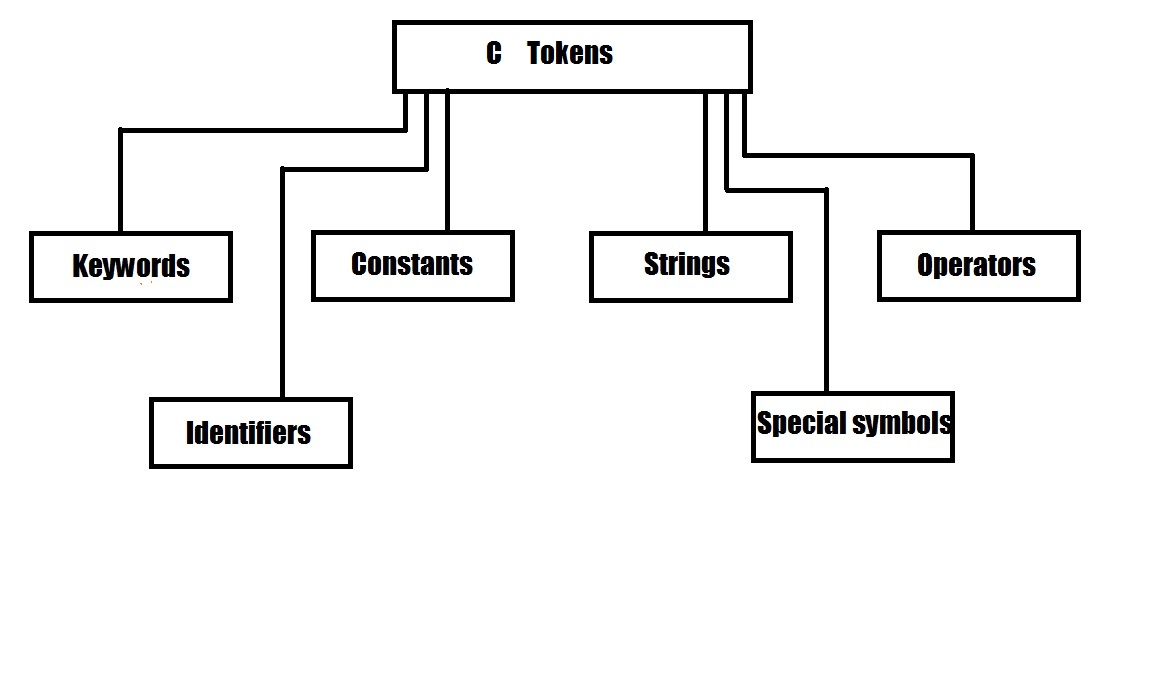
\v vertical tab

\\ backslash

\? question mark

\' quote

\" double quote



**Keywords in C**

Keywords are reserved words in C language. They have predicted meanings

1. **auto**
2. **break**
3. **case**
4. **char**
5. **const**
6. **continue,**
7. **default**
8. **do**
9. **double**
10. **Else**
11. **enum**
12. **extern**
13. **float**
14. **for**
15. **goto**
16. **if**
17. **int**
18. **long**
19. **register**
20. **return**
21. **short**
22. **signed**
23. **Size of**
24. **static**
25. **struct**
26. **switch**
27. **typedef**
28. **union**
29. **unsigned**
30. **void**
31. **volatile**
32. **While**

**OPERATORS**

**ARITHMETIC OPERATORS**

The basic operators for performing arithmetic are the same in many computer languages:

1. (+) addition
2. (-) subtraction
3. (\*) multiplication
4. (/) division
5. (%) modulus (remainder)

**ASSIGNMENT OPERATORS**

The most common assignment operator is”=”. This operator assigns the value in right side to the left side. For example:

X=4

**RELATIONAL OPERATORS**

Relational operator’s checks relationship between two operands. If the relation is true, it returns value 1 and if the relation is false, it returns value 0.

1. < ( less than )
2. <= (less than or equal to ),
3. > (greater than ),
4. >= ( greater than or equal to ),
5. = = ( equal to )
6. != (not equal to )

**LOGICAL OERATORS**

Logical operators are used to combine expressions containing relation operators. In C, there are 3 logical operators:

| **Operator** | **Meaning Of Operator** | **Example** |
| --- | --- | --- |
| && | Logial AND | If c=5 and d=2 then,((c==5) && (d>5)) returns false. |
| || | Logical OR | If c=5 and d=2 then, ((c==5) || (d>5)) returns true. |
| ! | Logical NOT | If c=5 then, !(c==5) returns false. |

### INCREMENT AND DECREMENT OPERATORS

In C, ++ and -- are called increment and decrement operators respectively. Both of these operators are unary operators, These operators can be prefix or postfix. but essentially with the prefix form the variable is changed *before* the value of the expression in which it appears is evaluated, and with the postfix form the variable is modified *afterwards*.

1. Prefix Increment and decrement operators (++a, --a)
2. Postfix Increment and decrement operators(a++, a--)

**CONDITIONAL OPERATOR**

The operator ?: is the conditional operator. It is used as

**variable 1 = expression 1 ? expression 2 : expression 3.**

Here expression 1 is a logical expression and expression 2 and expression 3 are expressions having numerical values. If expression 1 is true, value of expression 2 is assigned to variable 1 and otherwise expression3 is assigned.

**CONTROL STATEMENTS**

When we run a program, the statements are executed in the order in which they appear in the program. Also each statement is executed only once. But in many cases we may need a statement or a set of statements to be executed a fixed no of times or until a condition is satisfied. Also we may want to skip some statements based on testing a condition. For all these we use control statements.

***BRANCHING***

It is to execute one of several possible options depending on the outcome of a logical test, which is carried at some particular point within a program

**If Else Statement**

It is used to carry out one of the two possible actions depending on the outcome of a logical test.The else portion is optional.

**The syntax is**

**If (expression) \*/** if there is no else part

**{**

**statement1**

**}**

***Or***

**If (expression)**

**{**

**Statement 1**

**}**

**Else**

**{**

**Statement 2**

**}**

Here expression is a logical expression enclosed in parenthesis. If expression is true ,statement 1 or statement 2 is a group of statements ,they are written as a block using the braces { }

**nested if statement**

Within an if block or else block another if – else statement can come. Such statements are called nested if statements.

The syntax is

***If* (expression)**

***{***

**statement1**

**If (expression)**

***{***

**Statement2**

***}***

***Else***

***{***

**Statement3**

***}***

***}***

**Else**

**{**

**Statement 4**

**}**

**Ladder if statement**

Inorder to create a situation in which one of several courses of action is executed we use ladder – if statements.

The syntax is

**If (expression1)**

**{**

**Statement 1**

**}**

**Else If (expression2)**

**{**

**Statement 2**

**}**

**Else If (expression3)**

**{**

**Statement 3**

**}**

***--------------------***

***--------------------***

**LOOPING**

***The while statement***

This is to carry out a set of statements to be executed repeatedly until some condition is satisfied.

The syntax is:

**While (expression) statement**

The statement is executed so long as the expression is true. Statement can be simple or compound.

**Example** #include<stdio.h>

#incude <conio.h>

Void main()

{

Int n =20;

while(n > 0)

{

printf("\n %d",n);

n = n - 1;

}

Getch();

}

***do while statement***

This is also to carry out a set of statements to be executed repeatedly so long as a condition is true.

The syntax is:

**do**

**{**

**statement**

**} while(expression)**

**Example** #include<stdio.h>

#incude <conio.h>

Void main()

{

int i=1;

do

{

printf(“%d”,i);

++i;

}while(i<=10);

Getch();

}

***THE DIFFERENCE BETWEEN while loop AND do – while loop***

1) In the while loop the condition is tested in the beginning whereas in the other case it is done at the end.

2) In while loop the statements in the loop are executed only if the condition is true. Whereas in do – while loop even if the condition is not true the statements are executed at least once.

**for loop**

It is the most commonly used looping statement in C.

The Syntex is

**For(expression1;expression2;expression3)statement**

Here expression1 is to initialize some parameter that controls the looping action.expression2 is a condition and it must be true to carry out the action.expression3 is a unary expression or an assignment expression.

**Example:**

#include<stdio.h>

#incude <conio.h>

Void main()

{

int *i*;

for(*i*=1;*i*<=10;++*i*)

{

printf(“%d”,*i*);

}

Getch();

}

**Jumping**

**The break statement**

The break statement is used to terminate4 loop or to exit from a switch. It is used in for, while, do-while and switch statement.

The syntax is ***break;***

**Example 1:** A program to read the sum of positive numbers only

#include<stdio.h>

#incude <conio.h>

Void main()

{

int x, sum=0;

int n=1;

while(n<=10)

{

scanf(“%d”,&x);

if(x<0)

{

break;

}

sum=sum +x;

n= n+1;

}

printf(“%d”,sum);

getch();

}

**The continue statement**

It is used to bypass the remainder of the current pass through a loop. The loop does not terminate when continue statement is encountered, but statements after continue are skipped and proceeds to the next pass through the loop.

The syntax is ***continue;***

**Example :**

#include<stdio.h>

#incude <conio.h>

Void main()

{

int x, sum=0, n=1;

while(n<10)

{

scanf(“%d”,x);

if(x<0) continue;

sum+=x;

n=n+1;

}

printf(“%d”,sum);

getch();

}

**GO TO statement**

It is used to alter the normal sequence of program execution by transferring control to some other part of the program .The syntax is goto *label* ;

**Example :**

#include<stdio.h>

#incude <conio.h>

Void main()

{

int n=1,x,sum=0;

while(n<=10)

{

scanf(“%d” ,&x);

if(x<0)

{

goto error;

}

sum+=x;

++n;

}

error: printf(“\n the number is negative”);

}