C is a procedural language.

It is case-sensitive language.

HEADER FILES

Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processor *#include* statement. Standard header files are provided with each compiler, and cover a range of areas, string handling, mathematical, data conversion, printing and reading of variables.

To use any of the standard functions, the appropriate header file should be included. This is done at the beginning of the C source file. For example, to use the function *printf()* in a program, the line

#include <stdio.h>

should be at the beginning of the source file, because the definition for *printf()* is found in the file *stdio.h* All header files have the extension .h and generally reside in the /include subdirectory.

#include <stdio.h>

#include "mydecls.h"

The use of angle brackets <> informs the compiler to search the compilers include directory for the specified file. The use of the double quotes "" around the filename inform the compiler to search in the current directory for the specified file.

A C program, whatever its size, consists of functions and variables. A function contains statements that specify the computing operations to be done, and variables store values used during the computation.

All C programs consist of one or more functions. The only function that must be present is called main( ), which is the first function called when program execution begins.

For the C programming we use some constants, variables and keywords

Constant

A constant is an entity that does not change whereas a variable is an entity that may change.

const Int x=3;

There are 3 types of constants:

1) Primary Constant
2) Secondary Constant

Keywords

Keywords are the words whose meaning has already been explained to the C compiler. The keyword cannot be used as variable name because if we do so we are trying to assign a new meaning to the keyword.

There are 32 keywords in C

E.g. auto, int, char, double, break etc.

Variable

a variable is a named location in memory that is used to hold a value that can be modified by the program. All variables must be declared before they can be used. The general form of a declaration is

type variable_list;

Variables can be declared in three places: inside functions, in the definition of function parameters, and outside of all functions. These positions correspond to local variables, formal parameters, and global variables, respectively.

Local Variables

Variables that are declared inside a function are called *local variables*

```
void func1(void)
{
	int x;
	x = 10;
}
```

Formal Parameters

If a function is to use arguments, it must declare variables that will accept the values of the arguments. These variables are called the formal parameters of the function. They behave like any other local variables inside the function.

```
int abc(int c)
{
	int a=c*2;
	printf(“a=%d”,a);
	return 0;
}
```

Global Variables

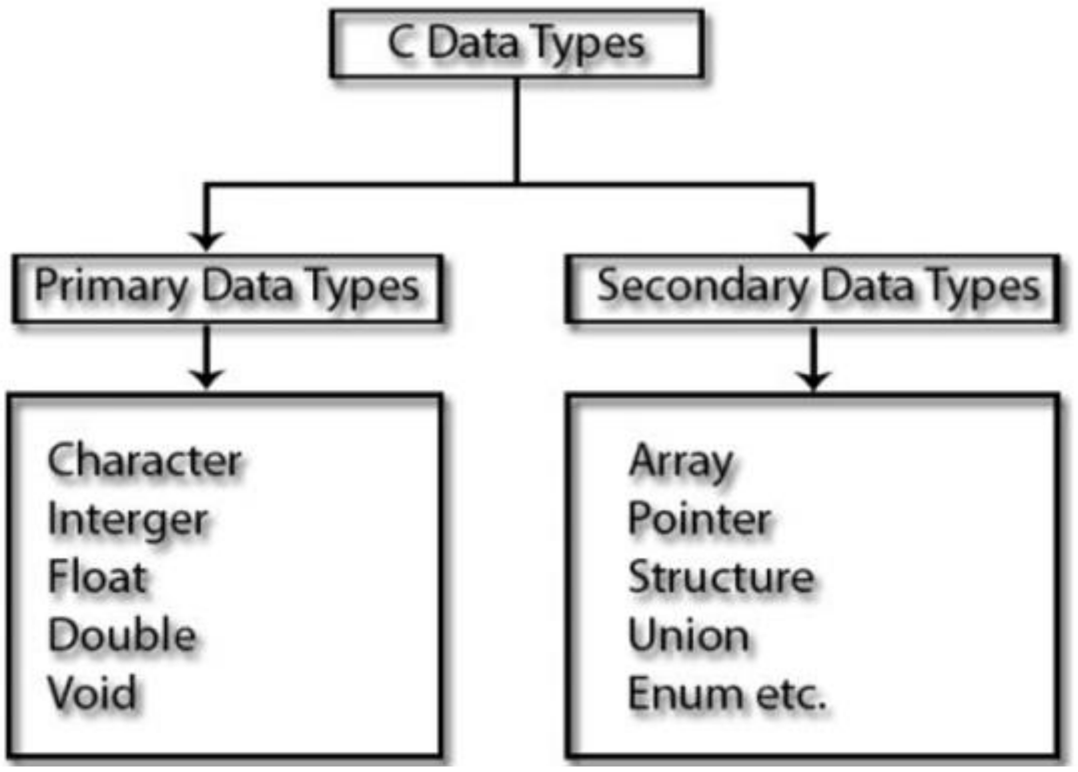
Unlike local variables, global variables are known throughout the program and may be used by any piece of code. Also, they will hold their value throughout the program's execution. You create global variables by declaring them outside of any function. Any expression may access them, regardless of what block of code that expression is in.

```
#include <stdio.h>
int count; /* count is global */
void func1(void);
void func2(void);
int main(void)
{
     count = 100;
     func1();
     return 0;
}
void func1(void)
{
     int temp;
     temp = count;
     func2();
     printf("count is %d", count); /* will print 100 */
}
void func2(void)
{
     int count;
     for(count=1;count<10;count++)
     putchar('.');
}
```

C has different data types for different types of data and can be broadly classified as
1) Primary data types
2) Secondary data types

Data Types in C

There are five atomic data types in the C subset: character, integer, floating-point, double floating-point, and valueless (char, int, float, double, and void, respectively). As we will see, all other data types in C are based upon one of these types.



Primary data types consist following data types.

Integer types:

Integers are whole numbers with a range of values, range of values are machine dependent. Generally an integer occupies 2 bytes memory space and its value range limited to -32768 to +32767 (that is, -2^{15} to $+2^{15}-1$). A signed integer use one bit for storing sign and rest 15 bits for number.

If you consider an integer having size of 4 byte (equal to 32 bits), it can take 2^{32} distinct states as: -2^{31}, $-2^{31}+1$, ...,-2, -1, 0, 1, 2, ..., $-2^{31}-2$, $-2^{31}-1$
Similarly, int of 2 bytes, it can take 216 distinct states from -215 to 215-1. If you try to store larger number than 231-1, i.e,+2147483647 and smaller number than -231, i.e, -2147483648, program will not run correctly

The format for declaring a variable is:

    type_specifier name;

Where

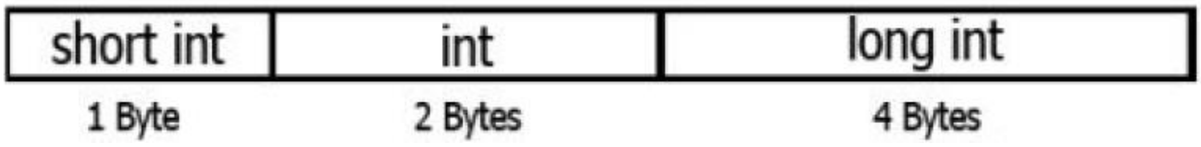
type_specifier is a reserved keyword (or combination of keywords) that specifies the data type , name is the name or 'identifier' for the variable

Syntax: int <variable name>;
like

    int num1;

```
        short int num2;
        long int num3;
```

Example: 5, 6, 100, 2500.
Integer Data Type Memory Allocation



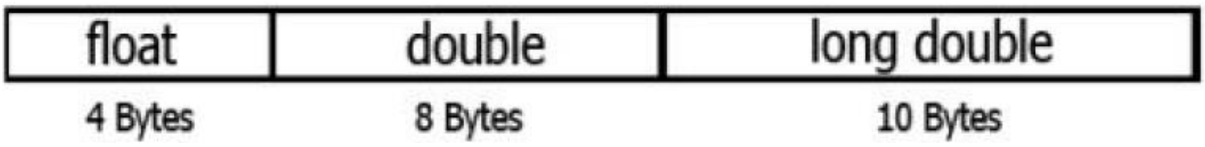
Floating Point Types:

The float data type is used to store fractional numbers (real numbers) with 6 digits of 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 and takes double space (8 bytes) than float. To extend the precision further we can use long double which occupies 10 bytes of memory space.

Syntax: float <variable name>; like
float num1;
double num2;
long double num3;
Example: 9.125, 3.1254.

Floating Point Data Type Memory Allocation

Variables of floating types can hold real values (numbers) such as: 2.34, -9.382 etc. Keywords either float or double is used for declaring floating type variable.



For example:

```
        float var2;
        double var3;
```

In C, floating values can be represented in exponential form as well. For example:

float var3=22.442e

Character Type:

Character type variable can hold a single character. As there are singed and unsigned int (either short or long), in the same way there are signed and unsigned chars; both occupy 1 byte each, but having different ranges. Unsigned characters have values between 0 and 255, signed characters have values from –128 to 127.

Syntax: char <variable name>; like
char ch = ‘a’;

The size of char is 1 byte. The character data type consists of ASCII characters. Each character is given a specific value. For example:

For, 'a', value =97
For, 'b', value=98
For, 'A', value=65
For, '&', value=33
For, '2', value=49

Character strings

Either C or C++ can represent constant strings using double quotes, such as "Hello World". The C language (as opposed to C++) uses arrays of single characters to store character strings as variables.

Boolean data

Boolean values are used for logic and decision making.

Special characters, as either single characters or within a string:
\n - New line character.
\t - Horizontal tab.
Etc.

Void Type

The void type has no values therefore we cannot declare it as variable as we did in case of integer and float.
The void data type is usually used with function to specify its type. Like in some C program we declared “main()” as void type because it does not return any value.

Enumeration

An enumeration is a set of named integer constants. Enumerations are common in everyday life. For example, an enumeration of the coins used in the United States is

penny, nickel, dime, quarter, half-dollar, dollar

The keyword **enum** signals the start of an enumeration type. The general form for enumerations is

enum *tag* { *enumeration list* } *variable_list*;

Here, both the tag and the variable list are optional. (But at least one must be present.) The following code fragment defines an enumeration called coin:

```
enum coin { penny, nickel, dime, quarter, half_dollar, dollar};
```

The enumeration tag name can be used to declare variables of its type. The following declares money to be a variable of type coin:

```
enum coin money;
```

Given these declarations, the following types of statements are perfectly valid:

```
money = dime;
if(money==quarter) printf("Money is a quarter.\n");
```

Each of the symbols stands for an integer value. As such, they can be used anywhere that an integer can be used. By default, the first is set to 0, and subsequent entries are incremented by 1 (values are generated for you)

Therefore,

```
printf("%d %d", penny, dime);
```
displays **0 2** on the screen.

We can specify the value of one or more of the symbols by using an initializer. Do this by following the symbol with an equal sign and an integer value. Symbols that appear after an initializer are assigned values greater than the preceding value. For example, the following code assigns the value of 100 to quarter:

```
enum coin { penny, nickel, dime, quarter=100, half_dollar, dollar};
```

Now, the values of these symbols are

penny	0
nickel	1
dime	2
quarter	100
half dollar	101
dollar	102

typedef

We can define new data type names by using the keyword typedef. We are not actually creating a new data type, but rather defining a new name for an existing type. This process can help make machine-dependent programs more portable.

The general form of the typedef statement is

```
typedef type newname;
```

where type is any valid data type, and newname is the new name for this type. The new name you define is in addition to, not a replacement for, the existing type name.
For example, you could create a new name for float by using

```
typedef float balance;
```

This statement tells the compiler to recognize balance as another name for float. Next, you could create a float variable using balance:

```
balance over_due;
```

Here, over_due is a floating-point variable of type balance, which is another word for float.
Now that balance has been defined, it can be used in another typedef. For example,

```
typedef balance overdraft;
```

tells the compiler to recognize overdraft as another name for balance, which is another name for float.
It helps in making the make your code easier to read and easier to port to a new machine.

Sample program illustrating each data type

```
#include < stdio.h >
main()
{
        int sum;
        float money;
        char letter;
        double pi;
        sum = 10; /* assign integer value */

        money = 2.21; /* assign float value */
        letter = 'A'; /* assign character value */
        pi = 2.01E6; /* assign a double value */
        printf("value of sum = %d\n", sum );
        printf("value of money = %f\n", money );
        printf("value of letter = %c\n", letter );
        printf("value of pi = %e\n", pi );
}
```

Output

```
value of sum = 10
value of money = 2.210000
value of letter = A
value of pi = 2.010000e+06
```

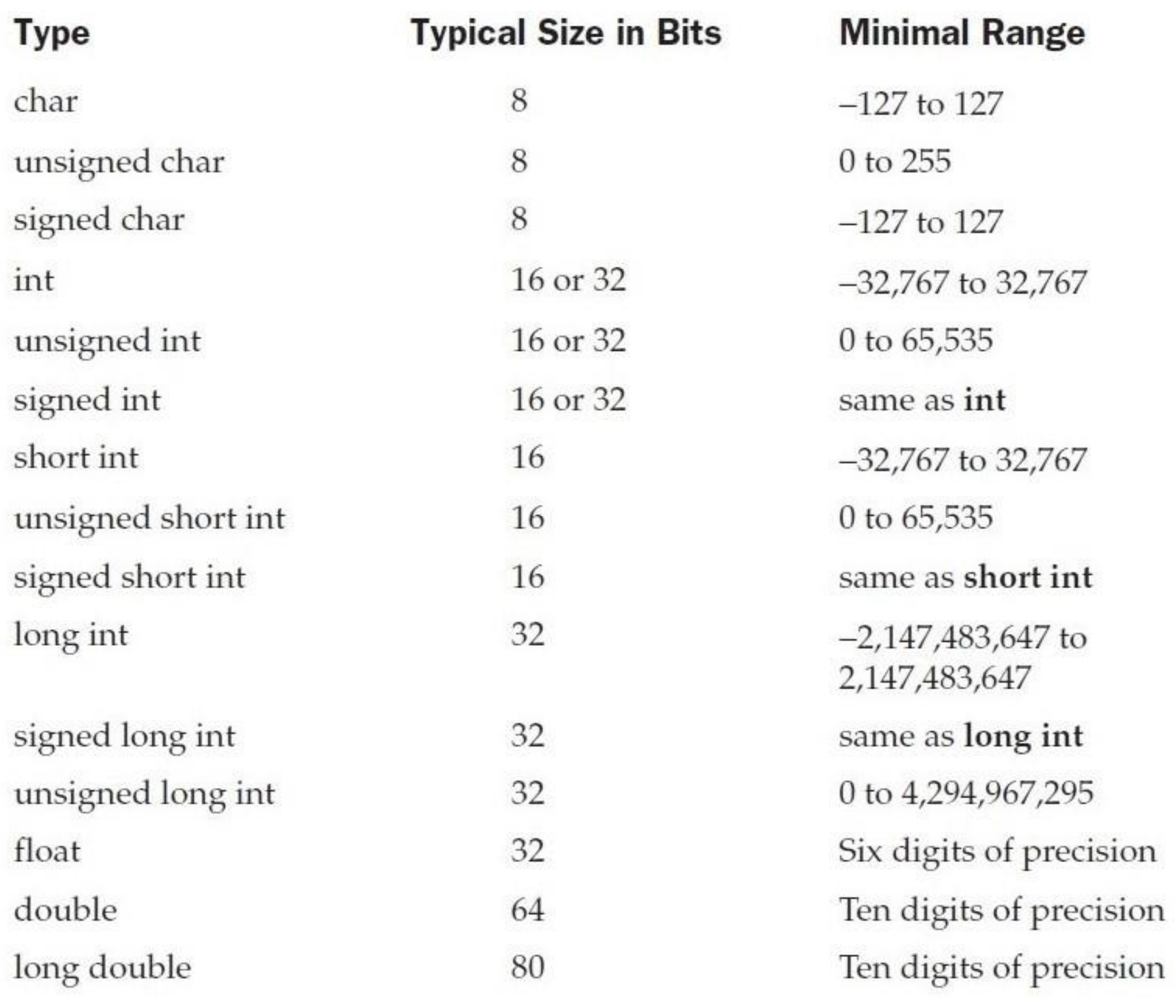
Modifiers

The data types explained above have the following modifiers.

short
long
signed
unsigned

The modifiers define the amount of storage allocated to the variable. The amount of storage allocated is not cast in stone. ANSI has the following rules:

short int <= int <= long int
float <= double <= long double



Type	Typical Size in Bits	Minimal Range
char	8	–127 to 127
unsigned char	8	0 to 255
signed char	8	–127 to 127
int	16 or 32	–32,767 to 32,767
unsigned int	16 or 32	0 to 65,535
signed int	16 or 32	same as **int**
short int	16	–32,767 to 32,767
unsigned short int	16	0 to 65,535
signed short int	16	same as **short int**
long int	32	–2,147,483,647 to 2,147,483,647
signed long int	32	same as **long int**
unsigned long int	32	0 to 4,294,967,295
float	32	Six digits of precision
double	64	Ten digits of precision
long double	80	Ten digits of precision

Secondary Data Types

An array in C language is a collection of similar data-type, means an array can hold value of a particular data type for which it has been declared. Arrays can be created from any of the C data-types int,...

Arrays:

We have seen all basic data types. In C language it is possible to make arrays whose elements are basic types. Thus we can make an array of 10 integers with the declaration.

int x[10];

The square brackets mean subscripting; parentheses are used only for function references. Array indexes begin at zero. Thus Array are special type of variables which can be used to store multiple values of same data type. Those values are stored and accessed using subscript or index. Arrays occupy consecutive memory slots in the computer's memory.

x[0], x[1], x[2], ..., x[9]

If an array has n elements, the largest subscript is n-1.
Multiple-dimension arrays are provided. The declaration and use look like:
int name[10] [20];

Here is a simple program

```
main( ) {
int n, c[100];
for(int i=0;i<10;i++)
{
        c[i] = n;
        n++;
}
        printf("length = %d\n", n);
}
```

C Instructions

There are three types of C Instructions:

1) Type Declaration Statement- Declares the type of variables used
2) Arithmetic Instruction – Perform arithmetic operations
3) Control Instruction – Controls the sequence of execution of various statements

Array Initialization

As with other declarations, array declarations can include an optional initialization Scalar variables are initialized with a single value while arrays are initialized with a list of values. The list is enclosed in curly braces

int array [8] = {2, 4, 6, 8, 10, 12, 14, 16};

The number of initializers cannot be more than the number of elements in the array but it can be less in which case, the remaining elements are initialized to 0.if you like, the array size can be inferred from the number of initializers by leaving the square brackets empty so these are identical declarations:

int array1 [8] = {2, 4, 6, 8, 10, 12, 14, 16};
int array2 [] = {2, 4, 6, 8, 10, 12, 14, 16};

An array of characters ie string can be initialized as follows:
char string[10] = "Hello";

Type conversion

When an operator has operands of different types, or a function gets a type different from that specified, type conversion occurs (if possible)

```
double x=1;
int y = x;

int x=1;
double y = x;
int x=365;
char c = x;
```

There are two types of type conversions:

1) **Implicit**: There are certain cases in which data will get automatically converted from one type to another:
   When data is being stored in a variable, if the data being stored does not match the type of the variable. The data being stored will be converted to match the type of the storage variable.
   When an operation is being performed on data of two different types. The "smaller" data type will be converted to match the "larger" type.
   For example, when an int is added to a double, the computer uses a double version of the int and the result is a double.

   The following example converts *the value of* nTotal to a double precision value before performing the division.

   ```
   average = nTotal / 3.0;
   ```

   Note that if the 3.0 were changed to a simple 3, then integer division would be performed, losing any fractional values in the result.
   When data is passed to or returned from functions.

2) **Explicit**: Data may also be expressly converted, using the typecast operator
   The following example converts *the value of* nTotal to a double precision value before performing the division. (nStudents will then be implicitly promoted, following the guidelines listed above.)

   ```
   average = ( double ) nTotal / nStudents;
   ```

   Note that nTotal itself is unaffected by this conversion

There are two other types of type conversions also present:

Narrowing conversion: It converts an object to a type that cannot include all of the values of the original type. e.g., float to int

Widening conversion: An object is converted to a type that can include at least approximations to all of the values of the original type. e.g., int to float

Qualifiers

Qualifiers alters the meaning of base data types to yield a new data type.

Size qualifiers:

Size qualifiers alters the size of basic data type. The keywords long and short are two size qualifiers.

For example:

long int i;

The size of int is either 2 bytes or 4 bytes but, when long keyword is used, that variable will be either 4 bytes of 8 bytes. If the larger size of variable is not needed then, short keyword can be used in similar manner as long keyword.

Sign qualifiers:

Whether a variable can hold only positive value or both values is specified by sign qualifiers. Keywords signed and unsigned are used for sign qualifiers.

unsigned int a;

// unsigned variable can hold zero and positive values only

It is not necessary to define variable using keyword signed because, a variable is signed by default.

Sign qualifiers can be applied to only int and char data types. For a int variable of size 4 bytes it can hold data from -231 to 231-1 but, if that variable is defined unsigned, it can hold data from 0 to 232 -1.

Constant qualifiers

Constant qualifiers can be declared with keyword const. An object declared by const cannot be modified.

const int p=20;

The value of *p* cannot be changed in the program.

Volatile qualifiers:

A variable should be declared volatile whenever its value can be changed by some external sources outside program. Keyword volatile is used to indicate volatile variable.

Arithmetic Statement

There are three types of arithmetic statement which are following;

1) Integer mode arithmetic statement – In this all the operands are either integer constants or integer variables.
   Eg int a,b,c;
   a = a+5;
   b=a*c-8;
2) Real mode arithmetic statement - In this all the operands are either real constants or real variables.
   Eg float a,b,c;
       a=a+5.5;
       b= a*b-3.8/9.7

3) Mixed mode arithmetic statement – In this some operand are real and some are integers
   float a,b,c;
   int d,e;
   a=b*c/9.5;
   d=e/9;

Order of Evaluation

Priority	Operators
1	**Postfix ++,--**
2	**Unary Operator, prefix ++,--**
3	*** / %**
4	**+ -**
5	**=**

Assignment Statement

Once we've declared a variable we can use it, but not until it has been declared - attempts to use a variable that has not been defined will cause a compiler error. Using a variable means storing something in it. You can store a value in a variable using:

name = value;
For example:
a=10;

stores the value 10 in the int variable a. What could be simpler? Not much, but it isn't actually very useful! Who wants to store a known value like 10 in a variable so you can use it later? It is 10, always was 10 and always will be 10. What makes variables useful is that you can use them to store the result of some arithmetic.

Consider four very simple mathematical operations: add, subtract, multiply and divide. Let us see how C would use these operations on two float variables a and b.

Add	a+b
Subtract	a-b
Multiply	a*b
Divide	a/b

Note that we have used the following characters from C's

character set:
+ for add
- for subtract
* for multiply

/ for divide

BE CAREFUL WITH ARITHMETIC!!! What is the answer to this simple calculation?
a=10/3
The answer depends upon how a was declared. If it was declared as type int the answer will be 3; if a is of type float then the answer will be 3.333.

Two points to note from the above calculation:
1. C ignores fractions when doing integer division!
2. When doing float calculations integers will be converted into float.

Conditional Operator
It is sometimes called ternary operator
Its general form is,
expression 1? Expression 2: expression 3

e.g. int a,b,c
a=0,b=1;
c= a>b?a:b

int a,b,c,d;
a=5,b=3,a=1;
d = (a>b)?(b>c)?b:c:a;

Overloaded Operators

Use of an operator for more than one purpose is called operator overloading
Some are common (e.g., + for int and string)