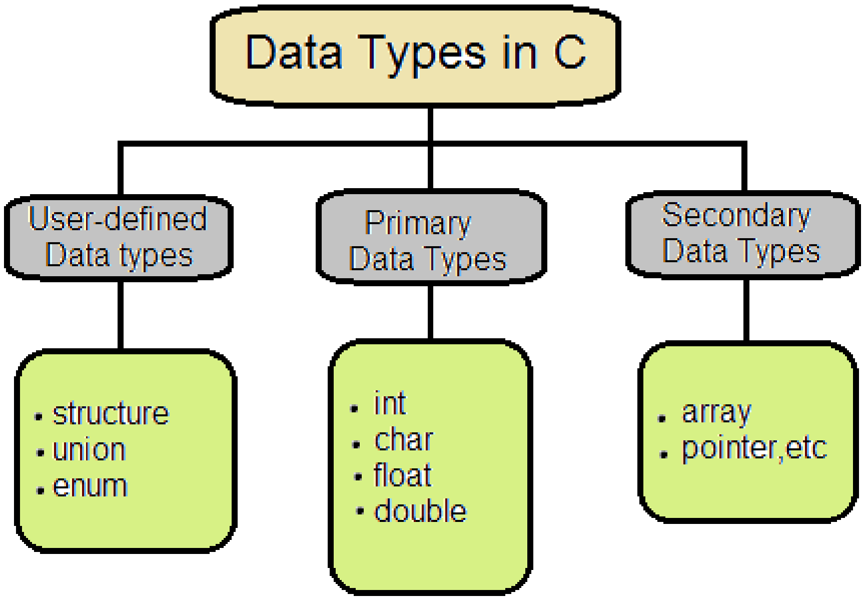
Data Type is the classification of the data that is taken as input, processed, and results in an output. It is how we categorize data according to its type.

There are three type of Data Types:

- ***Primary data types***
- ***Secondary data types***
- ***User-defined data types***

These data types are further subdivided into several other data types which you can find in the figure below:



Data types in C

1. Primary Data Types

Primary data types, also known as 'primitive data type' or 'fundamental data type', are the built-in data types that are provided by the programming language. It defines the most basic data like int, char, float, etc.

Primary data types could be of several types like an int can be unsigned int, short int, unsigned long int, etc. With such a wide range of classification and variety, the programmer has got many different data types to choose from as per the requirement and use it in their code along with the advantage of the secondary data types.

Here are the most commonly used primary data types:

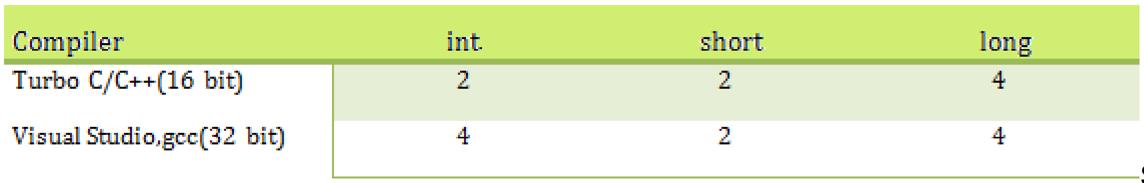
- boolean
- byte
- char
- short
- int
- long
- float
- double

'Void' is another primary data type that means 'no value'. It is usually used to define the type of return value in a function. A function with a void return type does not return any value.

1.1. Integer Data Type

Integer data types are used to define the variables taking integer values with or without constant values given to them. The most commonly used keyword or the data type used to define the integer type data is **'int'**. There are other data types like **'short'** and **'long'** used to define integer values but they have different ranges (we use them as per the requirement of the program i.e. if we are sure that our requirement is small and it will never go beyond the range of small we shall take small and likewise). In the tutorial constants, we saw that the range of the constants differed for different compilers.

Similarly, the size of the integer data type (in bytes) also differs for different compilers as shown in the table below:



Compiler	int	short	long
Turbo C/C++(16 bit)	2	2	4
Visual Studio,gcc(32 bit)	4	2	4

Size of int in different compilers

Note that the size of the given data types is in bytes.

The **short** integer can be used in places where small values and little storage space is required. It can boost up the runtime because it uses less space. It is declared by using the keyword **'short'** or **'short int'**.

The **long** integer gives us a long range or a bigger size compared to 'short' but it can cause our program to take more time for execution because of the storage size it offers. It is declared by using the keyword **'long'** or **'long int'**. The range for a long integer is –**2147483648** to +**2147483647**.

This is how we declare the integer variables:

```
int num,length;
```

```
short breadth;
short int height;
long int volume=0;
```

We have another classification of the integer data type: **signed** and **unsigned int**.

In case we need to take positive values only, then we can use unsigned int. Its range is 0 to 4294967295. This is because the leftmost bit is free and does not need to store the sign of the number. So, we get more storage space i.e. double on the positive side.

By default, the signed type is declared and we do not need to use signed. Signed int works like an int. Its range is the same as int.

1.2. Character Data Type

Character data types are used to define variables taking one character as its value. The keyword used for character data type is **'char'**. Here is how we declare character variables:

```
char ch, ch1='A', ch2=67;
```

Here, in ch1 variable, we store 'A', i.e., the binary equivalent of the ASCII value of A(=binary of decimal 65) gets stored. And ch2 variable stores the value 67(ASCII for 'C'). So these are actually two ways of initializing a character value.

Like integers, here we have **signed** and **unsigned** character values. The **signed char** is equivalent to **char**. For a signed char the range is **-128** to **+127**. Whereas, for an unsigned char the range is from **0** to **255**. Here's how we declare signed(declared as **char** above) and unsigned char values:

```
unsigned char ch;
char ch1=128;
```

Surprised to see why I put the value of ch1 as 128? Here's another thing about this data type. As mentioned before **char** has a range of **+127** but we have put the value **128** here. What is going to be the output?

What happens here is that once we reach the end of the range, the other side of the range is accessed. It goes back to -128 again. So in this case, when it comes across 128 it goes back to the beginning and accesses the char at the ASCII -128.

1.3. Float(and Double) Data Type

Float and double data types are used to define variables that take up a decimal value or an exponential value. The keyword used for float and double data type are **'float'** and **'double'** respectively.

- **Float** has a range of –**3.4e38** to **+3.4e38** and its size is **4 bytes**.
- **Double** has a range of **-1.7e308** to **+1.7e308** and its size is **8 bytes**.

- Another data type that is offered by programming languages is **'long double'** which has a range of –**1.7e4932** to **+1.7e4932** and its size is **10 bytes**. Here is how we declare float and double variables:

```
float length, area=0.0;
double radius, area=0.0;
```

---

2. Secondary Data Types

Secondary data types are basically derived from the primary data types. Let's have a look at a few secondary data types:

2.1. Arrays

An **array** is a collection of data of the same data type. These are declared under the same variable and are accessed using it. If we declare an integer array, all the values in the array have to be integers. Similarly, for a character array, all the elements of the array are characters and the same goes for double and every other data type. An array is declared as follows:

```
int a[50]; //Declaration
```

This array has an integer data type and can store 50 integer elements.

2.2. Pointers

A pointer contains the address of a variable in the program.
We declare the pointer as:

```
int *ip; //Declaration
```

A pointer declared as integer type stores the address of the integer type variable. Similarly, a pointer declared as char type stores the address of the character type variable and so on.

We will discuss about each one of them in detail in future posts.

---

3. User-defined data types

The user-defined data type defines the data in the way that the programmer chooses. Let's have a look at these commonly used user-defined data types:

3.1. Structures

It is a collection of variables of different data types represented by the same name. Unlike an array where we had to store all the data of the same type in the variable, here one can store data of different data types under the same variable name. It is mostly used to form records where different specifications need to be stored under the same name. The **struct** keyword is used to define a structure.

3.2.Union
Another data type that is very similar to structures. It allows the programmer to store data of different data types in the same memory location. A union can have multiple members but it can store only one member at a particular time. The keyword **union** is used to define a Union.

3.3. Enum
Enum or *Enumeration* is used to declare the variables and consists of integral constants. The keyword **enum** is used to define the enumeration data type.

Eg: enum identifier{element1, element2,……., elementn}; It assigns the value from **0** to **n** to the elements present inside the identifier sequentially.

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

There are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

Data Types	Memory Size	Range
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	

double	8 byte	
long double	10 byte	

Keywords in C

A keyword is a **reserved word**. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below:

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

Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

1. type variable_list;

The example of declaring the variable is given below:

1. **int** a;
2. **float** b;
3. **char** c;

Here, a, b, c are variables. The int, float, char are the data types.

We can also provide values while declaring the variables as given below:

```
int a=10,b=20;//declaring 2 variable of integer type
float f=20.8;
char c='A';
```

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Valid variable names:

```
int a;
int _ab;
int a30;
```

Invalid variable names:

```
int 2;
int a b;
int long;
```

Types of Variables in C

There are many types of variables in c:

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

Local Variable

A variable that is declared inside the function or block is called a local variable.

It must be declared at the start of the block.

```
void function1(){
int x=10;//local variable
}
```

You must have to initialize the local variable before it is used.

Global Variable

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

```
int value=20;//global variable
void function1(){
int x=10;//local variable
}
```

Static Variable

A variable that is declared with the static keyword is called static variable.

It retains its value between multiple function calls.

```
void function1(){
int x=10;//local variable
static int y=10;//static variable
x=x+1;
y=y+1;
printf("%d,%d",x,y);
}
```

If you call this function many times, the **local variable will print the same value** for each function call, e.g, 11,11,11 and so on. But the **static variable will print the incremented value** in each function call, e.g. 11, 12, 13 and so on.

Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

```c
void main(){
int x=10;//local variable (also automatic)
auto int y=20;//automatic variable
}
```

External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

myfile.h

```c
extern int x=10;//external variable (also global)
```

program1.c

```c
#include "myfile.h"
#include <stdio.h>
void printValue(){
    printf("Global variable: %d", global_variable);
}
```