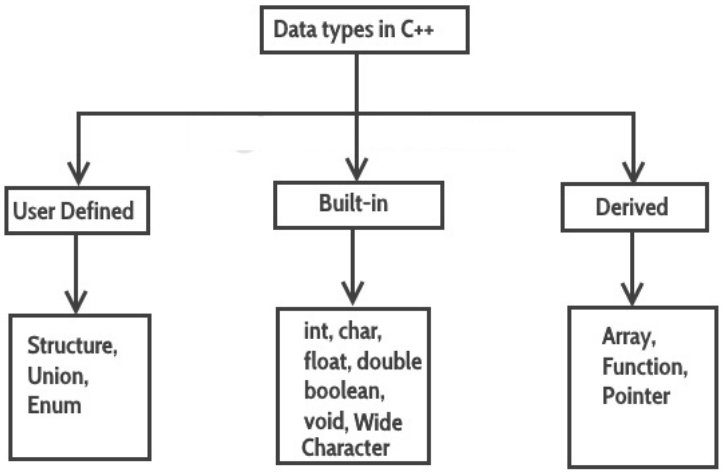
# Datatypes and Modifiers in C++

Let's start with Datatypes. They are used to define type of variables and contents used. Data types define the way you use storage in the programs you write. Data types can be of two types:

1. Built-in Datatypes
2. User-defined or Abstract Datatypes




## Basic Built in Datatypes in C++

| Type Names | Description | Size | Range |
|---|---|---|---|
| `char` | Single text character or small integer. Indicated with single quotes (`'a'`, `'3'`). | 1 byte | signed: -128 to 127<br>unsigned: 0 to 255 |
| `int` | Larger integer. | 4 bytes | signed: -2147483648 to 2147483647<br>unsigned: 0 to 4294967295 |
| `bool` | Boolean (true/false). Indicated with the keywords `true` and `false`. | 1 byte | Just `true` (1) or `false` (0). |
| `double` | "Doubly" precise floating point number. | 8 bytes | +/- 1.7e +/- 308 ( 15 digits) |

Example

```
char a = 'A';          // character type
int a = 1;             // integer type
float a = 3.14159;     // floating point type
double a = 6e-4;       // double type (e is for exponential)
```

## Other Built in Datatypes in C++

| | |
|---|---|
| `bool` | Boolean (True or False) |
| `void` | Without any Value |
| `wchar_t` | Wide Character |

## User-defined data types

We have three types of user-defined data types in C++
1. struct
2. union
3. enum

I have covered them in detail in separate tutorials. For now just remember that these comes under user-defined data types.

## Derived data types in C++

We have three types of derived-defined data types in C++
1. Array
2. Function
3. Pointer

# Modifiers in C++

In C++, special words(called **modifiers**) can be used to modify the meaning of the predefined built-in data types and expand them to a much larger set. There are four datatype modifiers in C++, they are:

1. `long`
2. `short`
3. `signed`
4. `unsigned`

The above-mentioned modifiers can be used along with built in datatypes to make them more precise and even expand their range.

Below mentioned are some important points you must know about the modifiers,

- **long** and **short** modify the maximum and minimum values that a data type will hold.
- A plain int must have a minimum size of **short**.
- Size hierarchy : `short int < int < long int`
- Size hierarchy for floating point numbers is : `float < double < long double`
- **long float** is not a legal type and there are no **short floating point** numbers.
- **Signed** types includes both positive and negative numbers and is the default type.
- **Unsigned**, numbers are always without any sign, that is always positive.

# C++ Variable

A variable is a name of 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:

```
int age = 14;
```

Here, `age` is a variable of the `int` data type, and we have assigned an integer value 14 to it.

# Types of variables

**int**: These type of of variables holds integer value.

There are three types of integer literals in C programming:

1. decimal (base 10)
2. octal (base 8)
3. hexadecimal (base 16)

**char**: holds character value like ‘c’, ‘F’, ‘B’, ‘p’, ‘q’ etc.

**bool**: holds boolean value true or false.

**double**: double-precision floating point value.

**float**: Single-precision floating point value.

## Rules for defining variables

- A variable name can only have alphabets, numbers and the underscore _.
- A variable name cannot begin with a number.
- Variable names cannot begin with an uppercase character.
- A variable name cannot be a keyword. For example, `int` is a keyword that is used to denote integers.
- A variable name can start with an underscore. However, it's not considered a good practice.

Valid variable names:

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

Invalid variable names:

```
int 4;
int x y;
int double;
```

## Scope of Variables

Any variable declared inside these curly braces have scope limited within these curly braces, if you declare a variable in main() function and try to use that variable outside main() function then you will get compilation error.

Now that we have understood what is scope. Lets move on to the types of variables based on the scope.

1. Global variable
2. Local variable

### Global Variable

A variable declared outside of any function (including main as well) is called global variable. Global variables have their scope throughout the program, they can be accessed anywhere in the program, in the main, in the user defined function, anywhere.

```
#include <iostream>
using namespace std;
// This is a global variable
char myVar = 'A';
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

## Local variable

Local variables are declared inside the braces of any user defined function, main function, loops or any control statements(if, if-else etc) and have their scope limited inside those braces.

```
#include <iostream>
using namespace std;

char myFuncn() {
// This is a local variable
char myVar = 'A';
}
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

## Can global and local variable have same name in C++?

```
#include <iostream>
using namespace std;
// This is a global variable
char myVar = 'A';
char myFuncn() {
    // This is a local variable
    char myVar = 'B';
    return myVar;
}
int main()
{
    cout <<"Funcn call: "<< myFuncn()<<endl;
    cout <<"Value of myVar: "<< myVar<<endl;
    myVar='Z';
    cout <<"Funcn call: "<< myFuncn()<<endl;
    cout <<"Value of myVar: "<< myVar<<endl;
    return 0;
}
```

```
Funcn call: B
Value of myVar: A
Funcn call: B
Value of myVar: Z
```

As you can see that when I changed the value of myVar in the main function, it only changed the value of global variable myVar because local variable myVar scope is limited to the function myFuncn().

## Some special types of variable

There are also some special keywords, to impart unique characteristics to the variables in the program. Following two are mostly used, we will discuss them in details later.

1. **Final** - Once initialized, its value cant be changed.
2. **Static** - These variables holds their value between function calls.

```
#include <iostream.h>
using namespace std;
int main()
{
    final int i=10;
    static int y=20;
}
```

## Escape Sequences

Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C++ programming. For example, newline (enter), tab, question mark, etc.

| Escape Sequences | Characters |
|---|---|
| \b | Backspace |
| \f | Form feed |

| | |
|---|---|
| `\n` | Newline |
| `\r` | Return |
| `\t` | Horizontal tab |
| `\v` | Vertical tab |
| `\\` | Backslash |
| `\'` | Single quotation mark |
| `\"` | Double quotation mark |
| `\?` | Question mark |
| `\0` | Null Character |

## C++ Constants

In C++, we can create variables whose value cannot be changed. For that, we use the `const` keyword. Here's an example:

```
const int LIGHT_SPEED = 152;
LIGHT_SPEED = 2500 // Error! LIGHT_SPEED is a constant.
```

A constant can also be created using the `#define` preprocessor directive.

# C++ Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc. **A list of 32 Keywords in C++ Language which are also available in C language are 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 |

**A list of 30 Keywords in C++ Language which are not available in C language are given below.**

| | | | | |
|---|---|---|---|---|
| asm | dynamic_cast | namespace | reinterpret_cast | bool |
| explicit | new | static_cast | false | catch |
| operator | template | friend | private | class |
| this | inline | public | throw | const_cast |
| delete | mutable | protected | true | try |
| typeid | typename | using | virtual | wchar_t |

## Differences between Identifiers and Keywords

| Identifiers | Keywords |
|---|---|
| Identifiers are the names defined by the programmer to the basic elements of a program. | Keywords are the reserved words whose meaning is known by the compiler. |
| It is used to identify the name of the variable. | It is used to specify the type of entity. |
| It can consist of letters, digits, and underscore. | It contains only letters. |

| | |
|---|---|
| It can use both lowercase and uppercase letters. | It uses only lowercase letters. |
| No special character can be used except the underscore. | It cannot contain any special character. |
| The starting letter of identifiers can be lowercase, uppercase or underscore. | It can be started only with the lowercase letter. |
| It can be classified as internal and external identifiers. | It cannot be further classified. |
| Examples are test, result, sum, power, etc. | Examples are 'for', 'if', 'else', 'break', etc. |