INDIAN INSTITUTE OF TECHNOLOGY ROORKEE

EEC-101 Programming with C++

Module-2.2 Constants, Variables, and Data Types

Ramanuja Panigrahi

- If you are asked you to retain the number 5 in your mental memory, and then you are asked to memorize also the number 2 at the same time.

- You have just stored two different values in your memory. Now, if you are asked to add 1 to the first number (i.e. 5), you should be retaining the numbers 6 (that is 5+1) and 2 in your memory.

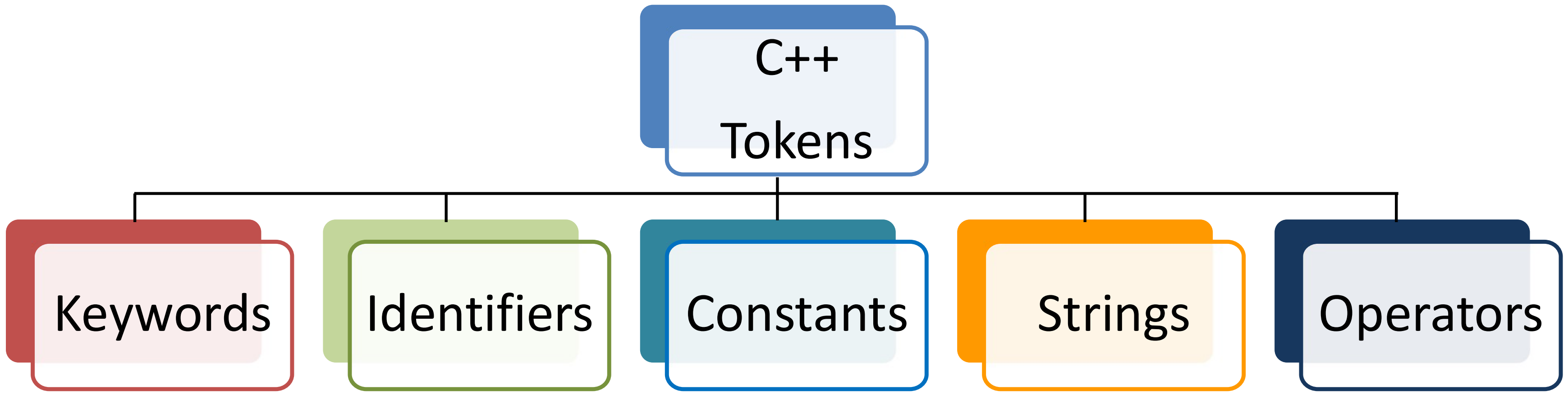
- The whole process that you have just done with your mental memory is similar to what a computer can do with two variables. The same process can be expressed in C++ with the following instruction set:
- a = 5;
- b = 2;
- a = a + 1;
- result = a - b;

(in this example we have yet not used the data types)

- Obviously, this is a very simple example since we have only used two small integer values, but consider that your computer can store millions of numbers like these at the same time and conduct sophisticated mathematical operations with them.

Tokens

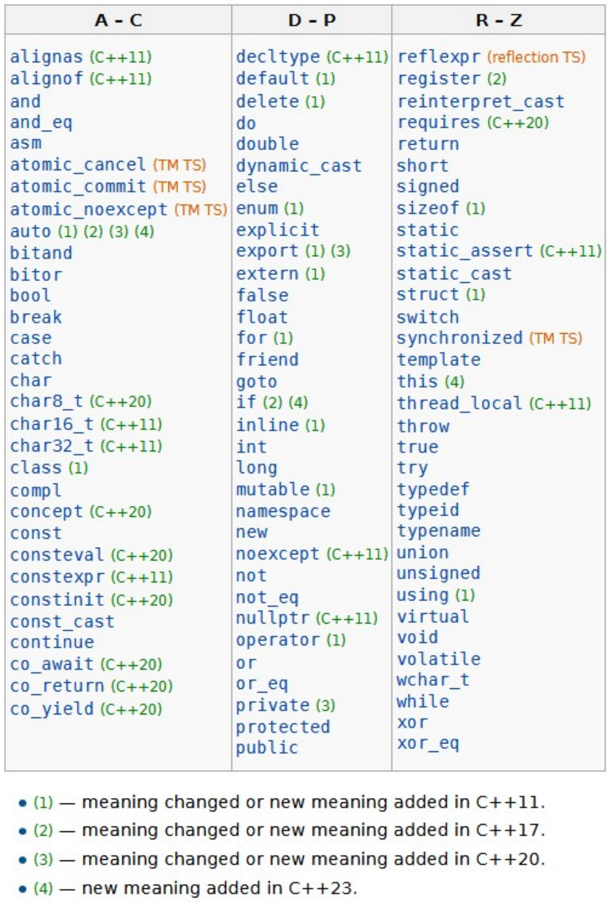
- A token in C++ can be defined as the smallest individual element of the C++ programming language that is meaningful to the compiler.
- When the compiler is processing the source code of a C++ program, each group of characters separated by white space is called a token.
- It is the basic component of a C++ program.



C++ Keywords

- Keywords are part of the vocabulary of a programming language
- In most languages, the keywords are also reserved. This means that you, the programmer, cannot redefine their meaning and you can't use them in a way in which there were not intended.

C++ Keywords



A - C	D - P	R - Z
alignas (C++11)	decltype (C++11)	reflexpr (reflection TS)
alignof (C++11)	default (1)	register (2)
and	delete (1)	reinterpret_cast
and_eq	do	requires (C++20)
asm	double	return
atomic_cancel (TM TS)	dynamic_cast	short
atomic_commit (TM TS)	else	signed
atomic_noexcept (TM TS)	enum (1)	sizeof (1)
auto (1) (2) (3) (4)	explicit	static
bitand	export (1) (3)	static_assert (C++11)
bitor	extern (1)	static_cast
bool	false	struct (1)
break	float	switch
case	for (1)	synchronized (TM TS)
catch	friend	template
char	goto	this (4)
char8_t (C++20)	if (2) (4)	thread_local (C++11)
char16_t (C++11)	inline (1)	throw
char32_t (C++11)	int	true
class (1)	long	try
compl	mutable (1)	typedef
concept (C++20)	namespace	typeid
const	new	typename
consteval (C++20)	noexcept (C++11)	union
constexpr (C++11)	not	unsigned
constinit (C++20)	not_eq	using (1)
const_cast	nullptr (C++11)	virtual
continue	operator (1)	void
co_await (C++20)	or	volatile
co_return (C++20)	or_eq	wchar_t
co_yield (C++20)	private (3)	while
	protected	xor
	public	xor_eq

- (1) — meaning changed or new meaning added in C++11.
- (2) — meaning changed or new meaning added in C++17.
- (3) — meaning changed or new meaning added in C++20.
- (4) — new meaning added in C++23.

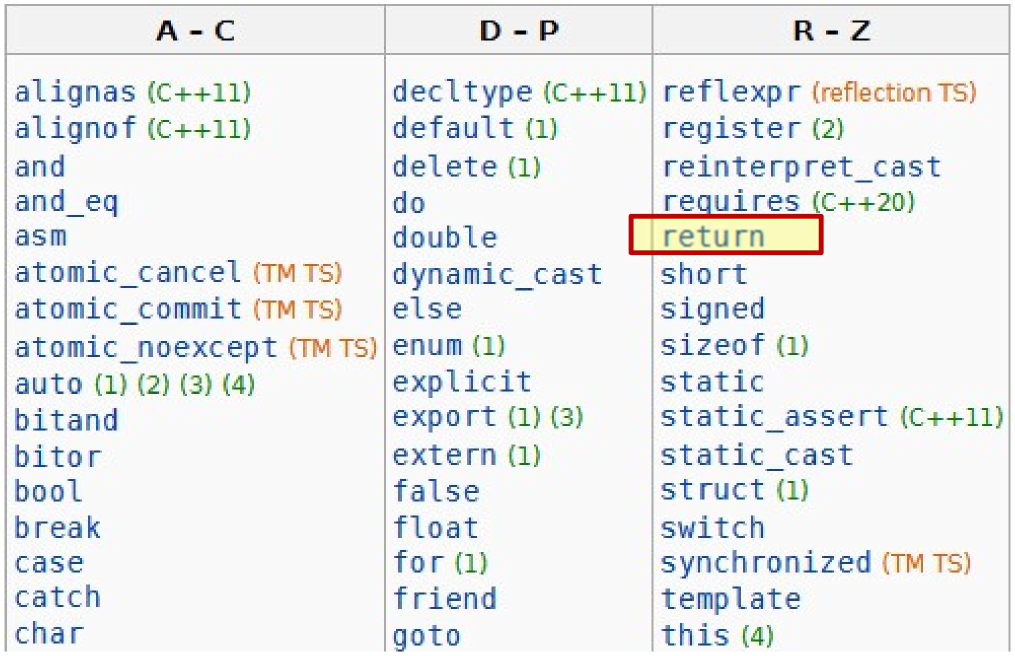
https://en.cppreference.com/w/cpp/keyword

C++ (C++20) (92 keywords)
Java (SE 17 LTS) (67 keywords)
MATLAB (R2020a) (20 keywords)
Python 3 (3.10) (38 keywords)

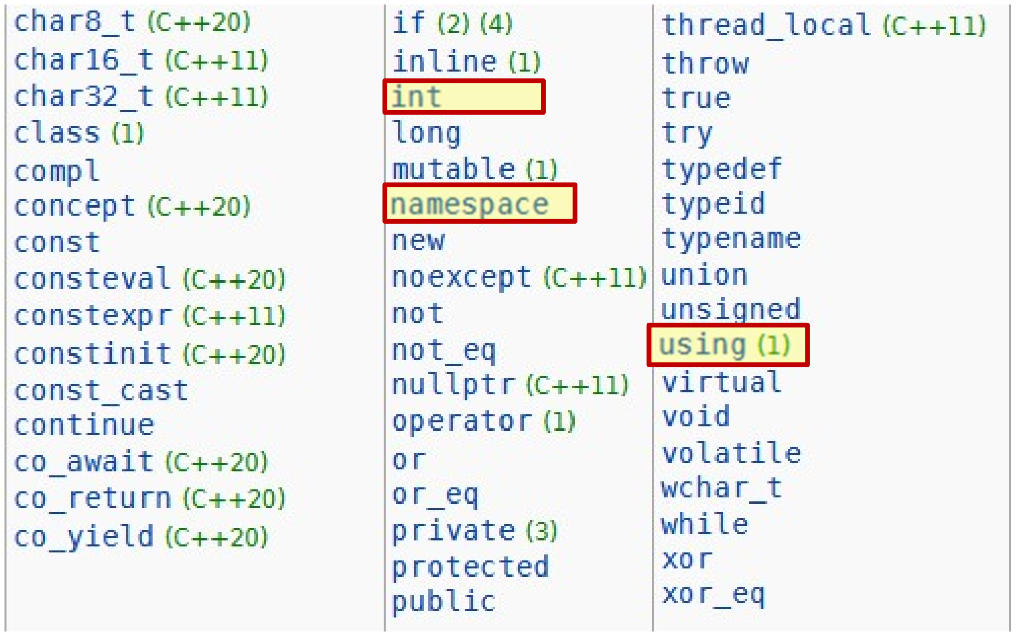
- C++ has lots of keywords
- Now in general, the more keywords that a language has, the more complex the programming language grammar.
- However, there are many keywords in C++ that are rarely used.

There's no need to memorize all these keywords.
We'll use the keywords as we need them.

What we have used



A - C	D - P	R - Z
alignas (C++11)	decltype (C++11)	reflexpr (reflection TS)
alignof (C++11)	default (1)	register (2)
and	delete (1)	reinterpret_cast
and_eq	do	requires (C++20)
asm	double	return
atomic_cancel (TM TS)	dynamic_cast	short
atomic_commit (TM TS)	else	signed
atomic_noexcept (TM TS)	enum (1)	sizeof (1)
auto (1) (2) (3) (4)	explicit	static
bitand	export (1) (3)	static_assert (C++11)
bitor	extern (1)	static_cast
bool	false	struct (1)
break	float	switch
case	for (1)	synchronized (TM TS)
catch	friend	template
char	goto	this (4)



char8_t (C++20)	if (2) (4)	thread_local (C++11)
char16_t (C++11)	inline (1)	throw
char32_t (C++11)	int	true
class (1)	long	try
compl	mutable (1)	typedef
concept (C++20)	namespace	typeid
const	new	typename
consteval (C++20)	noexcept (C++11)	union
constexpr (C++11)	not	unsigned
constinit (C++20)	not_eq	using (1)
const_cast	nullptr (C++11)	virtual
continue	operator (1)	void
co_await (C++20)	or	volatile
co_return (C++20)	or_eq	wchar_t
co_yield (C++20)	private (3)	while
	protected	xor
	public	xor_eq

Identifiers

- Within a program names are used to designate variables and functions.
- Identifiers are used for naming variables, functions, and arrays.

Rules for creating identifiers:

1. A name contains a series of letters, numbers, or underscore characters (_).
2. C++ is case sensitive; that is, upper- and lowercase letters are different.
3. The first character must be a letter or underscore
4. There are no restrictions on the length of a name and all the characters in the name are significant
5. C++ keywords are reserved and cannot be used as names

- Things to know
  - Neither spaces nor punctuation marks or symbols can be part of an identifier.
  - In no case they can begin with a digit.

Naming Conventions:

1. The C++ compiler uses internal names that begin with one or two underscores followed by a capital letter. To avoid confusion with these names, avoid use of the underscore at the beginning of a name.
2. To improve the readability of your programs you should choose longer and more self-explanatory names.

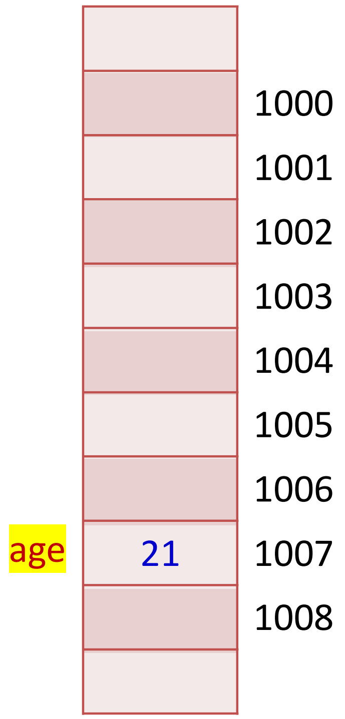
For example, score_math or scoreMath represents an identifier related to how much mark is scored in math.

Variables

Variables from a conceptual point of view

- Random access memory, or RAM, is a contiguous block of storage used by the computer to store information.
- Ram can be thought of as having memory cells where each cell has an associated location.
- If we had to program using the specific memory locations, it wouldn't be a lot of fun, and we'd likely have a lot of programmer errors.

EX- Imagine saying something like move 21 to memory location 1007.

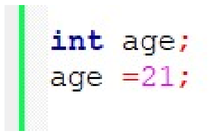


- In this example, you can see that the memory location 1007 has been associated
- with the name age.
- In computing this is called a binding, and we can move 21 to age.
- This allows us not to have to worry about what exact memory location age is associated with. If we run the program again, age will likely be associated with a different memory address. And that's okay since we never knew about it in the first place.

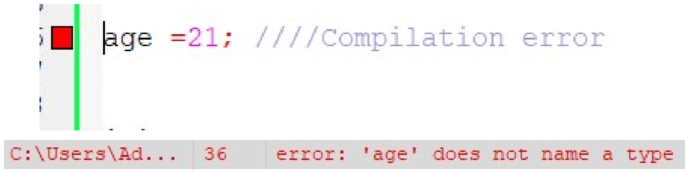
What is a variable?

- A variable is an abstraction for a memory location.
- Each variable needs an identifier that distinguishes it from the others
- Variables have
  - Type- Decides the category (integer, real number, string, …)
  - Value- Decides the contents (5, -2.15, “Hello World”,…)

- A variable must be declared before it can be used.



```
int age;
age =21;
```



```
age =21; ////Compilation error
```

```
C:\Users\Ad...   36   error: 'age' does not name a type
```

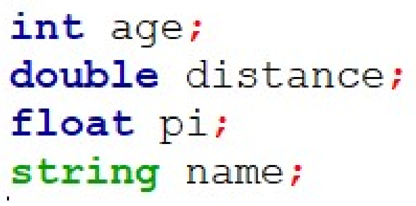
- The value of a variable may vary

Declaring and Initializing Variables

// Declaring a single variable

type variable_name;

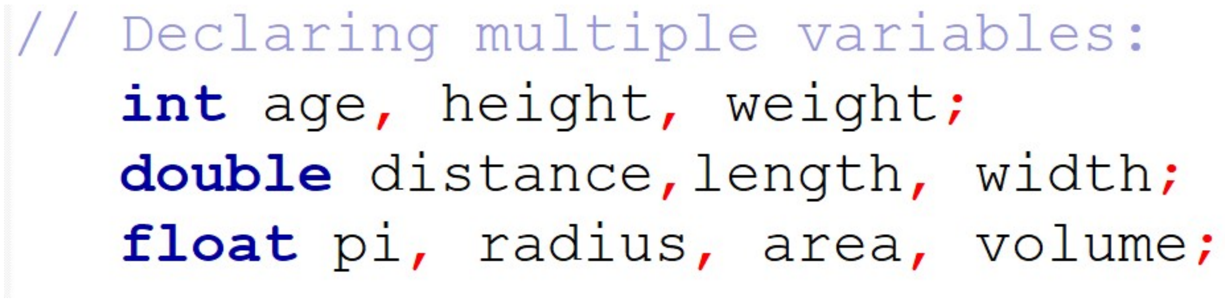
Example:



```
int age;
double distance;
float pi;
string name;
```

// Declaring multiple variables:

type variable1_name, variable2_name, variable3_name;



```
// Declaring multiple variables:
   int age, height, weight;
   double distance,length, width;
   float pi, radius, area, volume;
```

Rules for Variable names

- The variable name is a sequence of one or more letters, digits, or underscore, for example character_123
- In C++, using uppercase or lowercase letters matters. Variables Age and age mean different things to the compiler.
- White space, punctuation symbols, or other characters are not permitted to denote variable names.
- Variable names cannot be keywords or any reserved words of the C++ programming language.
- A variable name must begin with a letter or underscore.

- Naming Variables

Legal	Illegal
Age	int
age	&age
_age	2024_myage
My_age_in_2024	My age
INT	Age+1
	cout

- Best Practices:
  - Be consistent my_age_2024 vs. myAge2024
  - Avoid underscore at the beginning
  - Use meaningful names
  - Never use variables before initializing them
  - Declare variables close to when you need.

- C++ is a case-sensitive language.

- For example, the variable name **TIGER** differs from the variable name **Tiger or tiger or tIger.**

Storing values in memory

Assignment statement

- stores the value of right hand side expression into a variable.

```
int x, y;
x = 5;        //Value 5 gets stored in a memory location which is referred by the name x.
y = m + 2;    //Value 7 gets stored in y
```

- '=' is called assignment operator (binary)
- LHS is a data object name (Lvalue – a value that can be modified)
- It can be a dereferenced pointer or a reference also
- More modern name is modifiable Lvalue
- RHS is an expression (Rvalue)

Initializing variables

```
int age; // uninitialized

int age=21; // C like initialization
int age (0);// constructor type initialization
int age {21};// C++ 11 list initialization syntax
```

Always use this

- We will use the C++ 11 list initialization, as it has certain advantages.
- Variable refers to a memory location. But how much memory is allotted depends on the data type of the variable.

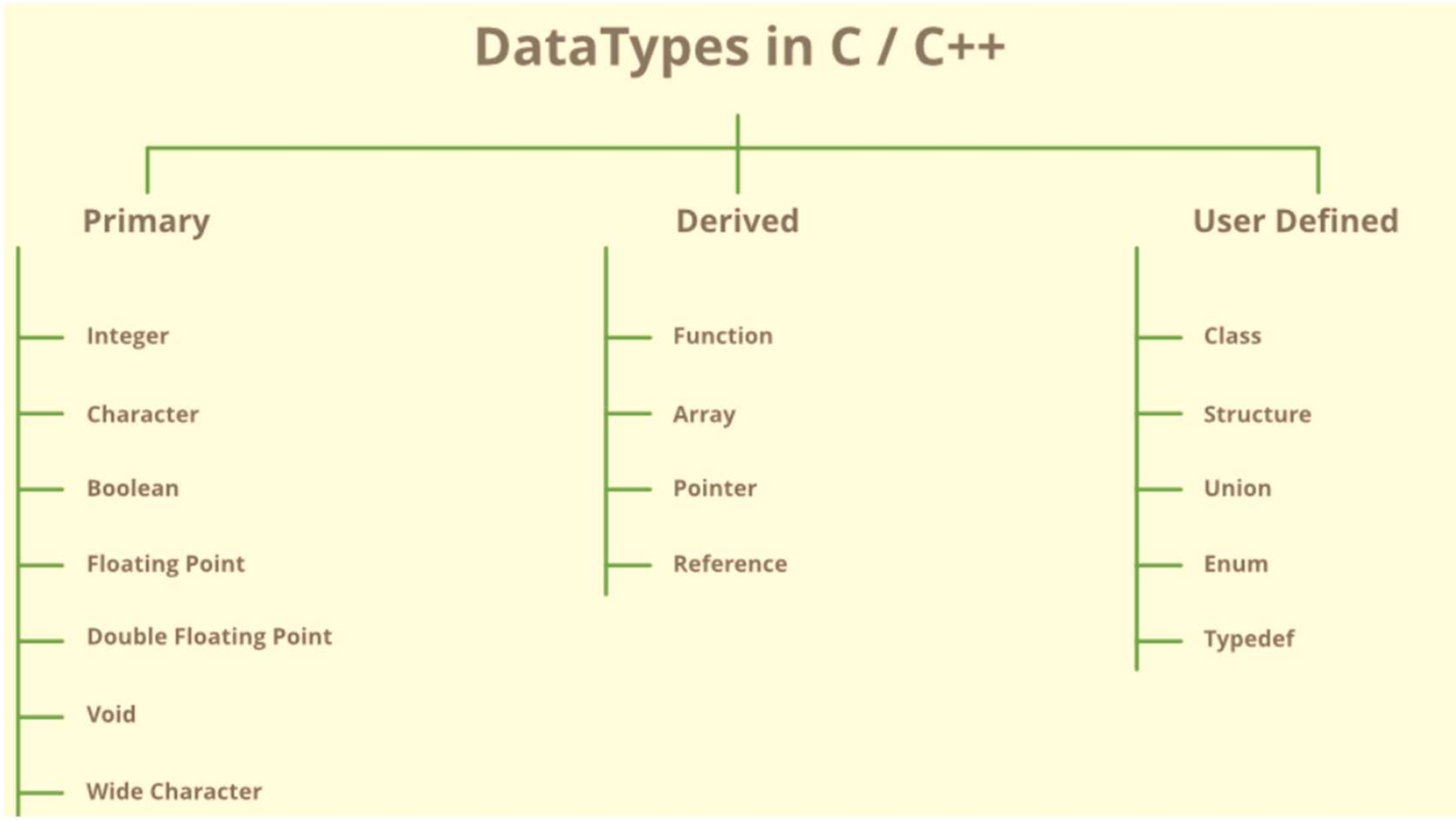
Data Types

- When programming, variables are stored in computer's memory
- Computer has to know what kind of data we want to store in them
- Occupies different amounts of memory to store a simple number a single letter or a large number
- The memory in our computers is organized in bytes.

- A byte is the minimum amount of memory that we can manage in C++.
- A byte can store a relatively small amount of data: one single character or a small integer (generally an integer between 0 and 255).
- In addition, the computer can manipulate more complex data types that come from grouping several bytes, such as long numbers or non-integer numbers

C++ Data Types

- Data types specify the size and types of values to be stored.



Data Types in C++

- **C++ supports the following data types:**
  - **Primary** or **Built-in** or **Fundamental data type**
  - **Derived data types**
  - **User-defined data types**

- **1. Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. Primitive data types available in C++ are:
  - Integer
  - Character
  - Boolean
  - Floating Point
- 2. **Derived Data Types**: Data types that are derived from the primitive datatypes are referred to as Derived Data Types. These can be of four types namely:
  - Function
  - Array
  - Pointer
  - Reference

Data types

- 3. **Abstract or User-Defined Data Types**: Abstract or User-Defined data types are defined by the user itself. Like, defining a class in C++ or a structure. C++ provides the following user-defined datatypes:

  - Class
  - Structure
  - Union
  - Enumeration
  - Typedef defined Datatype

C++ Primitive data types

- **Primitive Data Types**: These data types are built-in or predefined data types and can be used directly by the user to declare variables. example: int, char, float, bool, etc.
- Data types implemented directly by C++ language
    - Character Types
    - Integer Types
    - Floating point Type
    - Boolean Type

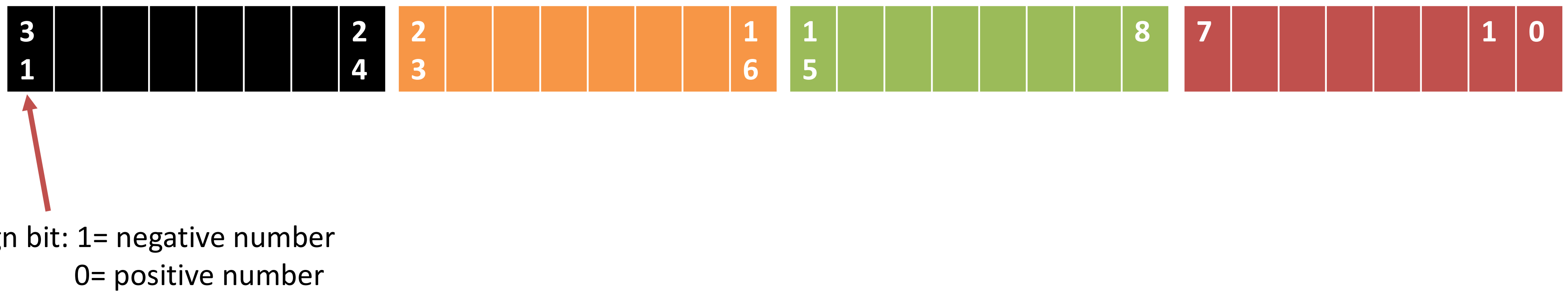
- Size and precision are often compiler-dependent.
    – #include <climits>

- Size is expressed in Bits
- More bits can represent more values
- More bit needs more storage

Size (in bits)	Representable Values	
8	256	2^8
16	65,536	2^{16}
32	4,294,967,296	2^{32}
64	18,446,744,073,709,551,615	2^{64}

Integer type

- **Integer**: The keyword used for integer data types is **int**. Integers typically require 4 bytes of memory space and range from -2147483648 to 2147483647.



Data Type Modifiers

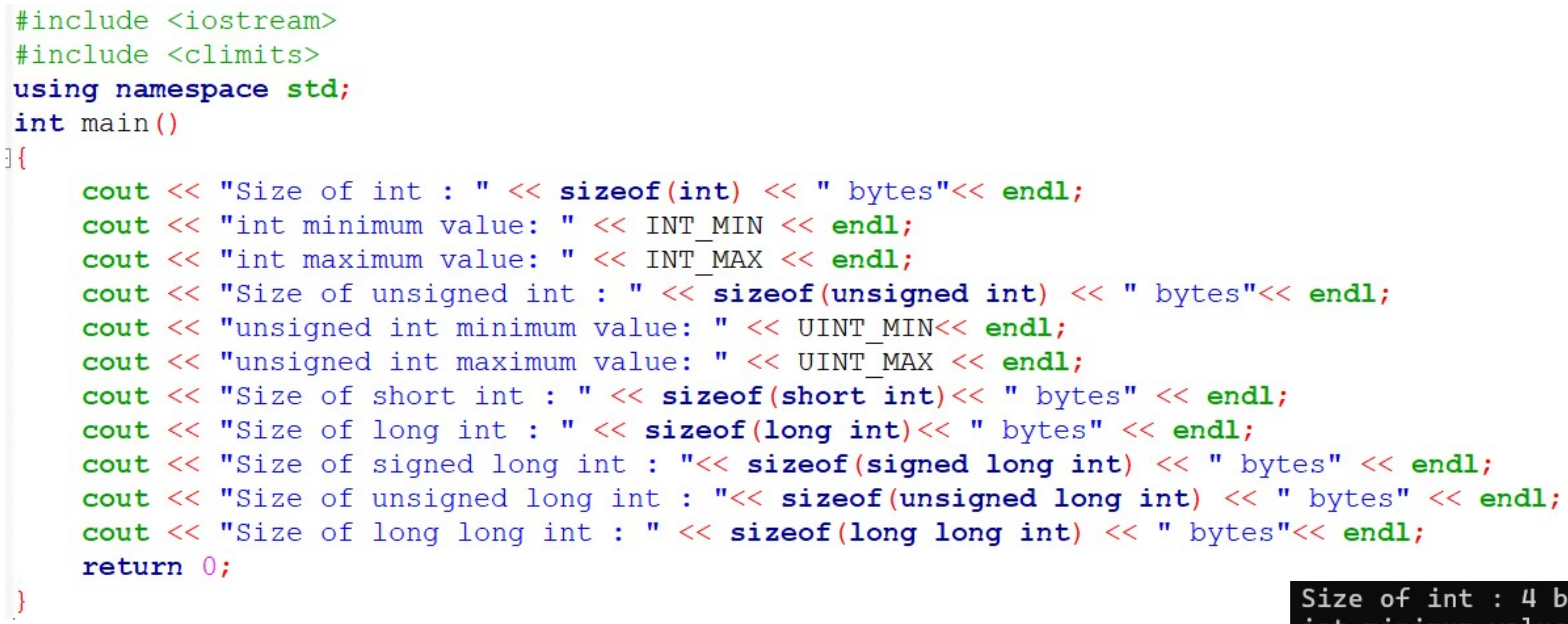
- **Datatype Modifiers**
- As the name suggests, datatype modifiers are used with built-in data types to modify the length of data that a particular data type can hold.

- Data type modifiers available in C++ are:
  - **Signed**
  - **Unsigned**
  - **Short**
  - **Long**

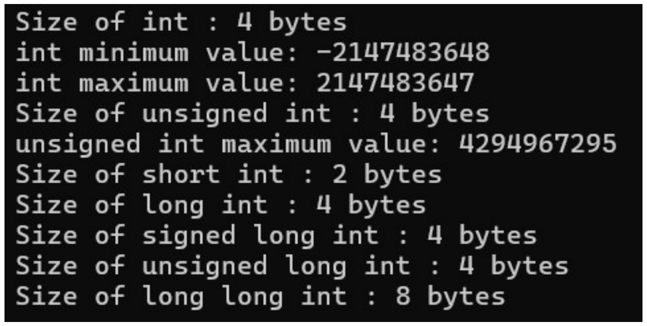
Modified Integer

Data Type	Size (in bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	$-(2^{63})$ to $(2^{63})-1$
unsigned long long int	8	0 to 18,446,744,073,709,551,615

Example



```
#include <iostream>
#include <climits>
using namespace std;
int main()
{
    cout << "Size of int : " << sizeof(int) << " bytes"<< endl;
    cout << "int minimum value: " << INT_MIN << endl;
    cout << "int maximum value: " << INT_MAX << endl;
    cout << "Size of unsigned int : " << sizeof(unsigned int) << " bytes"<< endl;
    cout << "unsigned int minimum value: " << UINT_MIN<< endl;
    cout << "unsigned int maximum value: " << UINT_MAX << endl;
    cout << "Size of short int : " << sizeof(short int)<< " bytes" << endl;
    cout << "Size of long int : " << sizeof(long int)<< " bytes" << endl;
    cout << "Size of signed long int : "<< sizeof(signed long int) << " bytes" << endl;
    cout << "Size of unsigned long int : "<< sizeof(unsigned long int) << " bytes" << endl;
    cout << "Size of long long int : " << sizeof(long long int) << " bytes"<< endl;
    return 0;
}
```

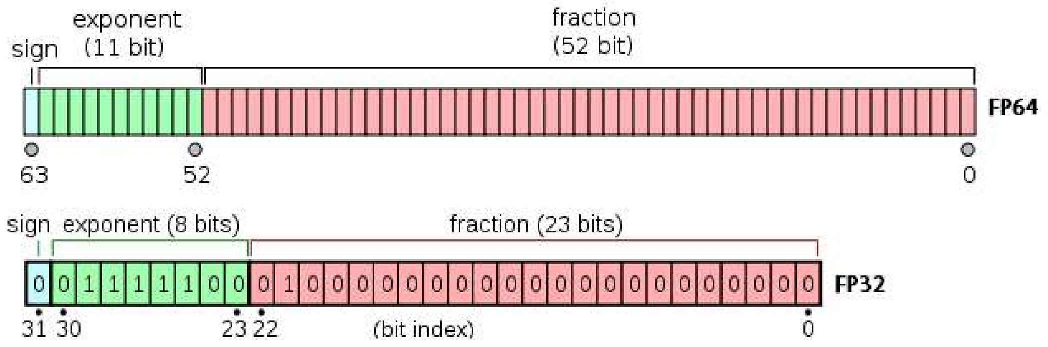


```
Size of int : 4 bytes
int minimum value: -2147483648
int maximum value: 2147483647
Size of unsigned int : 4 bytes
unsigned int maximum value: 4294967295
Size of short int : 2 bytes
Size of long int : 4 bytes
Size of signed long int : 4 bytes
Size of unsigned long int : 4 bytes
Size of long long int : 8 bytes
```

Floating Point Type

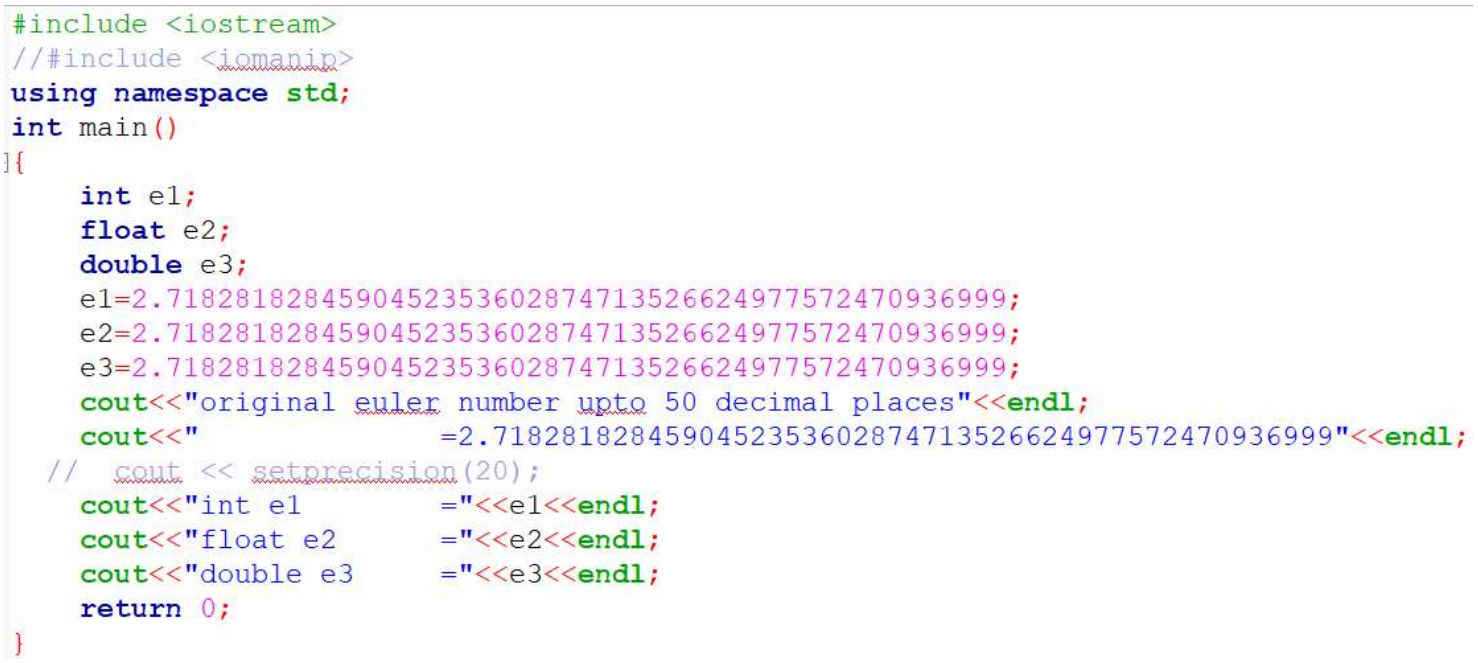
- Floating Point:
  - Floating Point data type is used for storing single-precision floating-point values or decimal values.
  - The keyword used for the floating-point data type is float.
  - Float variables typically require 4 bytes of memory space.
- Double Floating Point:
  - Double Floating Point data type is used for storing double-precision floating-point values or decimal values.
  - The keyword used for the double floating-point data type is double.
  - Double variables typically require 8 bytes of memory space.

float	4	-3.4×10^{38} to 3.4×10^{38}
double	8	-1.7×10^{308} to 1.7×10^{308}
long double	12	-1.1×10^{4932} to 1.1×10^{4932}

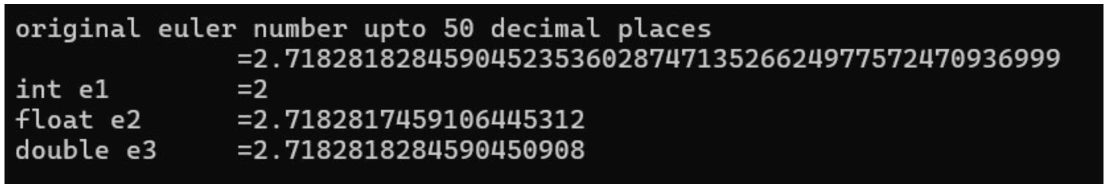


https://www.h-schmidt.net/FloatConverter/IEEE754.html

https://www.wikihow.com/Convert-a-Number-from-Decimal-to-IEEE-754-Floating-Point-Representation



```cpp
#include <iostream>
//#include <iomanip>
using namespace std;
int main()
{
    int e1;
    float e2;
    double e3;
    e1=2.71828182845904523536028747135266249775724709369999;
    e2=2.71828182845904523536028747135266249775724709369999;
    e3=2.71828182845904523536028747135266249775724709369999;
    cout<<"original euler number upto 50 decimal places"<<endl;
    cout<<"              =2.71828182845904523536028747135266249775724709369999"<<endl;
  //  cout << setprecision(20);
    cout<<"int e1        ="<<e1<<endl;
    cout<<"float e2      ="<<e2<<endl;
    cout<<"double e3     ="<<e3<<endl;
    return 0;
}
```

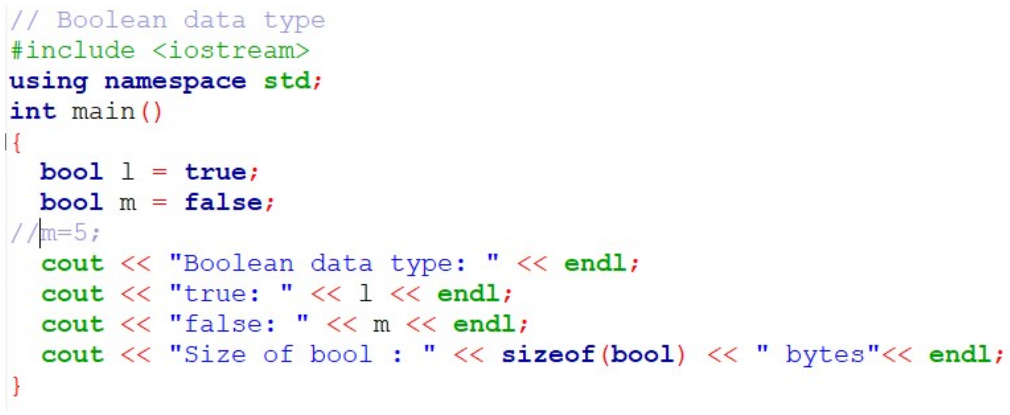


```
original euler number upto 50 decimal places
              =2.71828182845904523536028747135266249775724709369999
int e1        =2
float e2      =2.7182817459106445312
double e3     =2.7182818284590450908
```

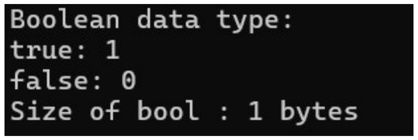
Boolean

- **Boolean**: Boolean data type is used for storing Boolean or logical values.
  - A Boolean variable can store either *true* or *false*.
  - The keyword used for the Boolean data type is **bool**.
    - These values are stored as integers 0 and 1
    - zero =false
    - Non-zero=true

Example



```
// Boolean data type
#include <iostream>
using namespace std;
int main()
{
  bool l = true;
  bool m = false;
//m=5;
  cout << "Boolean data type: " << endl;
  cout << "true: " << l << endl;
  cout << "false: " << m << endl;
  cout << "Size of bool : " << sizeof(bool) << " bytes"<< endl;
}
```



```
Boolean data type:
true: 1
false: 0
Size of bool : 1 bytes
```

Character Type

- **Character**: Character data type is used for storing characters.
  - The keyword used for the character data type is **char**.
  - Characters typically require 1 byte of memory space and range from -128 to 127 or 0 to 255.

- A character type is an integral type whose variables represent characters like the letter ‘A’ or the digit ‘8’.

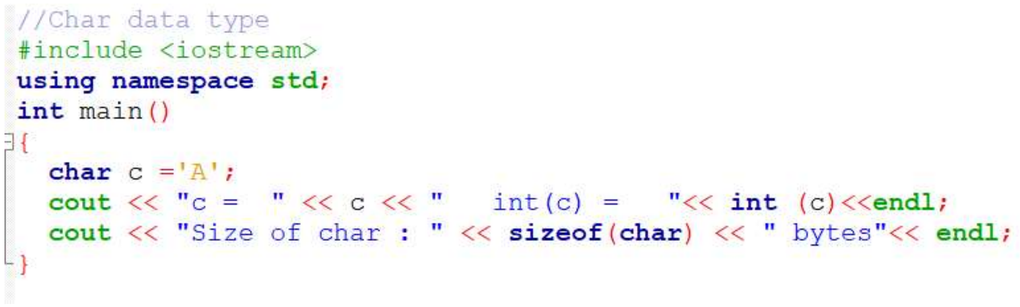
- Character literals are delimited by the apostrophe (’)

- Character values are stored as integers char c =‘A’

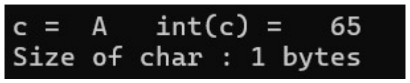
cout << “c = “ << c << “   int(c) =   “<< int (c)<<endl;

Output

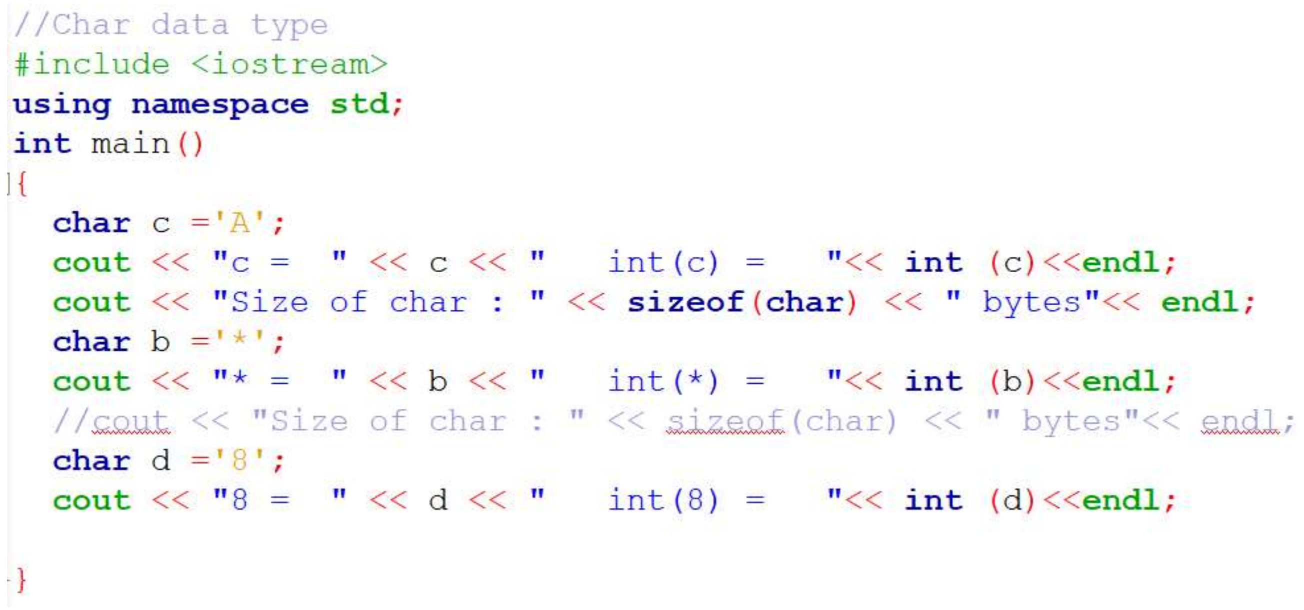
c= A,   int(c) = 65



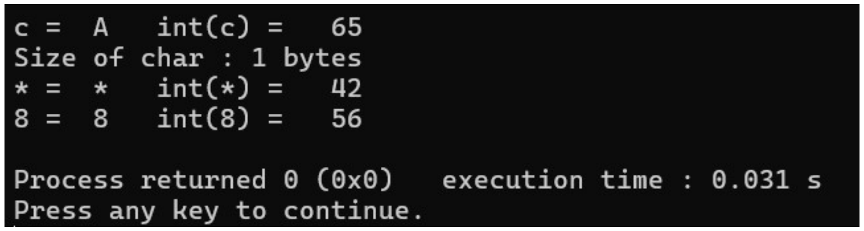
```
//Char data type
#include <iostream>
using namespace std;
int main()
{
   char c ='A';
   cout << "c =  " << c << "   int(c) =   "<< int (c)<<endl;
   cout << "Size of char : " << sizeof(char) << " bytes"<< endl;
}
```



```
c =  A   int(c) =   65
Size of char : 1 bytes
```



```cpp
//Char data type
#include <iostream>
using namespace std;
int main()
{
  char c ='A';
  cout << "c =   " << c << "   int(c) =   "<< int (c)<<endl;
  cout << "Size of char : " << sizeof(char) << " bytes"<< endl;
  char b ='*';
  cout << "* =   " << b << "   int(*) =   "<< int (b)<<endl;
  //cout << "Size of char : " << sizeof(char) << " bytes"<< endl;
  char d ='8';
  cout << "8 =   " << d << "   int(8) =   "<< int (d)<<endl;

}
```

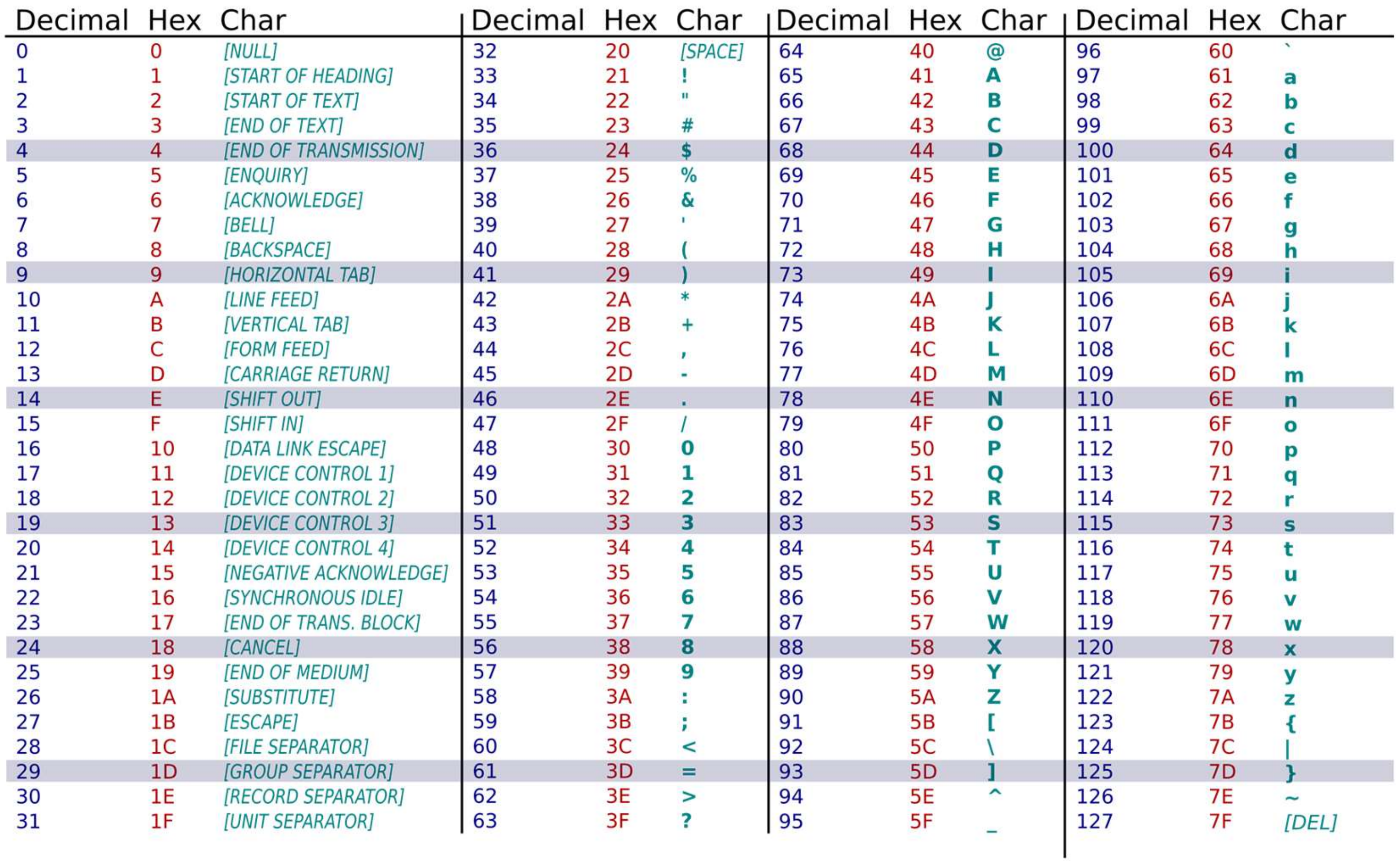


```
c =  A   int(c) =   65
Size of char : 1 bytes
* =  *   int(*) =   42
8 =  8   int(8) =   56

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

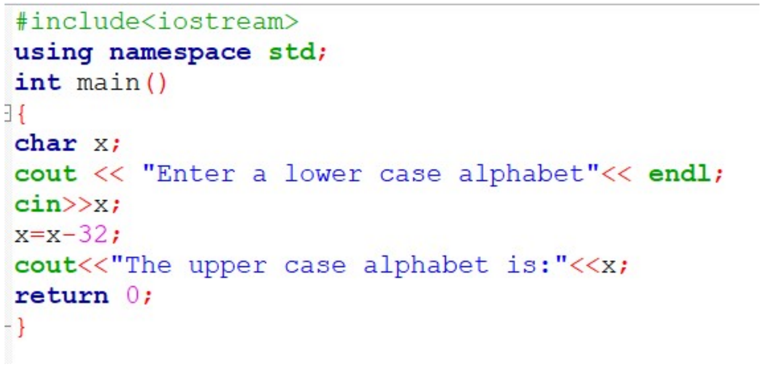
Character Representation

- ASCII Codes
  - American Standard Code for Information Interchange
  - Standard encoding scheme used to represent characters in binary format on computers
  - 7-bit encoding, so 128 characters can be represented
    - With n digits, 2^n unique numbers (from 0 to 2^n-1) can be represented. If n=7, 128 (=2^7) numbers can be represented 0-127.
  - 0 to 31 (& 127) are "control characters" (cannot print)
    - Ctrl-A or ^A is 1, ^B is 2, etc.
    - Used for screen formatting & data communication
  - 32 to 126 are printable

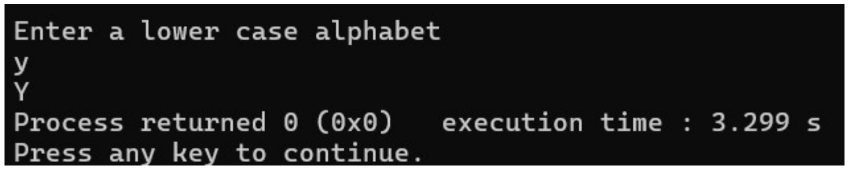
ASCII Table



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]



```
#include<iostream>
using namespace std;
int main()
{
char x;
cout << "Enter a lower case alphabet"<< endl;
cin>>x;
x=x-32;
cout<<"The upper case alphabet is:"<<x;
return 0;
}
```



```
Enter a lower case alphabet
y
Y
Process returned 0 (0x0)   execution time : 3.299 s
Press any key to continue.
```

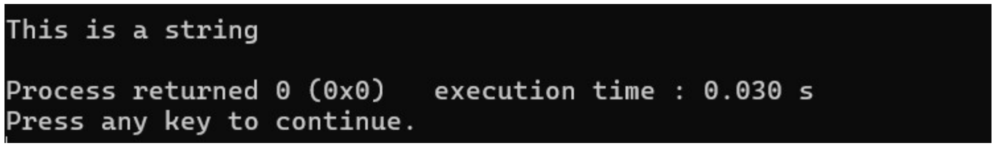
Wide Character:

- Wide character data type is also a character data type but this data type has a size greater than the normal 8-bit data type.
- Represented by wchar_t.
- It is generally 2 or 4 bytes long.

Strings

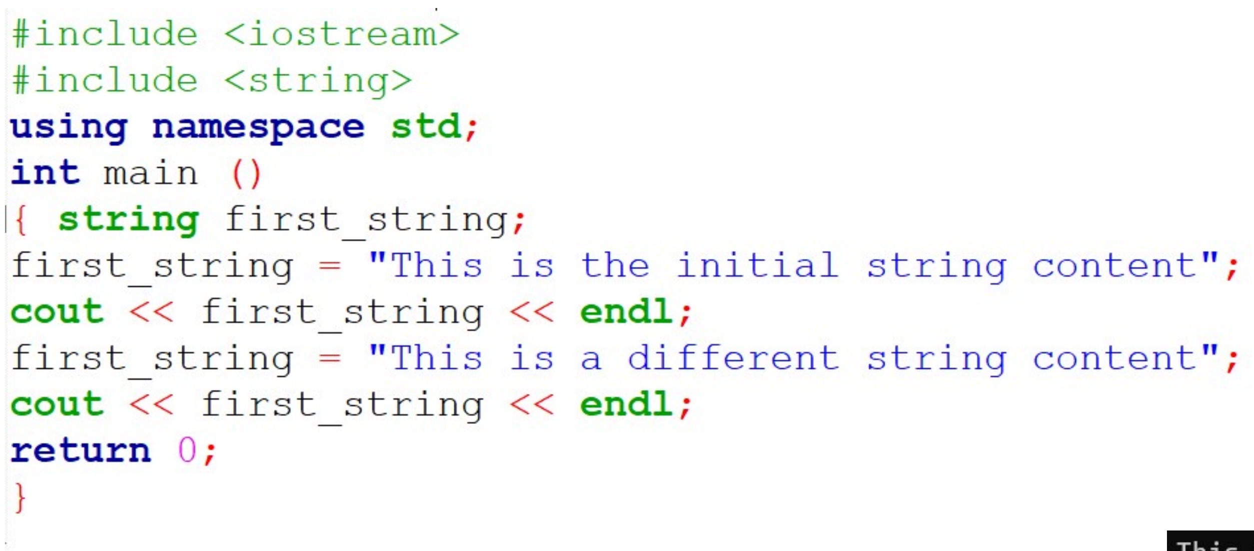
- Variables that can store non-numerical values that are longer than one single character are known as strings.
- The C++ language library provides support for strings through the standard string class.
- This is not a fundamental type, but it behaves in a similar way as fundamental types do in its most basic usage.
- A first difference with fundamental data types is that in order to declare and use objects (variables) of this type we need to include an additional header file in our source code: <string> and have access to the std namespace (which we already had in all our previous programs thanks to the using namespace statement).

```cpp
// my first string
#include <iostream>
#include <string>
using namespace std;
int main ()
{
string  mystring = "This is a string";
cout << mystring;
cout<<endl;
return 0;
}
```

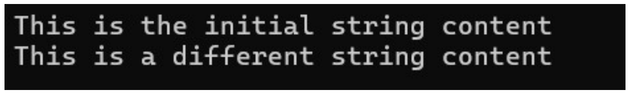


```
This is a string

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

- Strings can also perform all the other basic operations that fundamental data types can, like being declared without an initial value and being assigned values during execution:

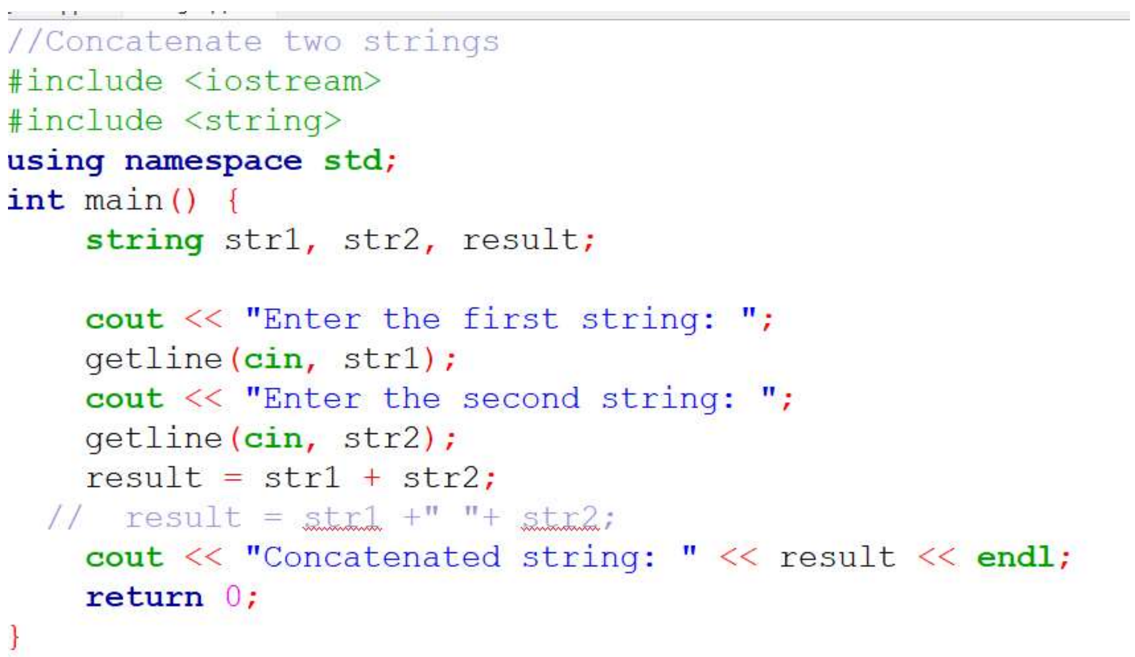


```
#include <iostream>
#include <string>
using namespace std;
int main ()
{ string first_string;
first_string = "This is the initial string content";
cout << first_string << endl;
first_string = "This is a different string content";
cout << first_string << endl;
return 0;
}
```



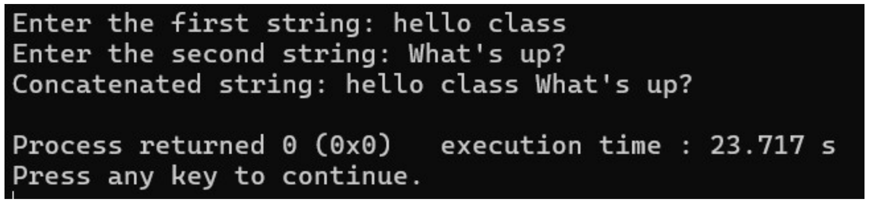
```
This is the initial string content
This is a different string content
```

Concatenation



```
//Concatenate two strings
#include <iostream>
#include <string>
using namespace std;
int main() {
    string str1, str2, result;

    cout << "Enter the first string: ";
    getline(cin, str1);
    cout << "Enter the second string: ";
    getline(cin, str2);
    result = str1 + str2;
  //  result = str1 +" "+ str2;
    cout << "Concatenated string: " << result << endl;
    return 0;
}
```

Concatenation:
 the action of linking things together in a series, or the condition of being linked in such a way.

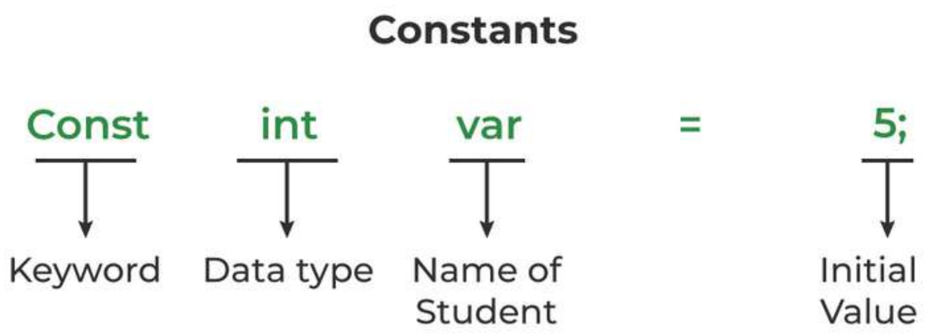


```
Enter the first string: hello class
Enter the second string: What's up?
Concatenated string: hello class What's up?

Process returned 0 (0x0)   execution time : 23.717 s
Press any key to continue.
```

Constants

- Constants are expressions with a fixed value.
- Similar to Variables, constants also
  - Have names
  - Occupy storage
  - Usually typed in
  - However, their value cannot change once declared.

- The type of constant can be an integer constant, a floating pointer constant, a string constant, or a character constant.
- In C++ language, the const keyword is used to define the constants.



Literal Constants

- Literals are the most obvious kind of constants. They are used to express particular values within the source code of a program
- for example

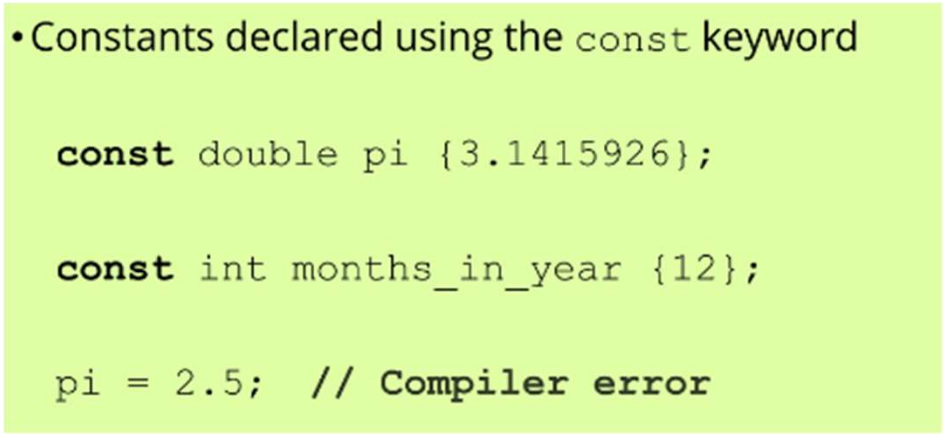
```
a = 9;
```

- the 9 in this piece of code was a literal constant.

Literal constants can be divided in Integer Numerals, Floating-Point Numerals, Characters, Strings and Boolean Values.

```
x = 12;
y = 1.56;
name = “Frank”;
middle_initial = ‘J’;
```

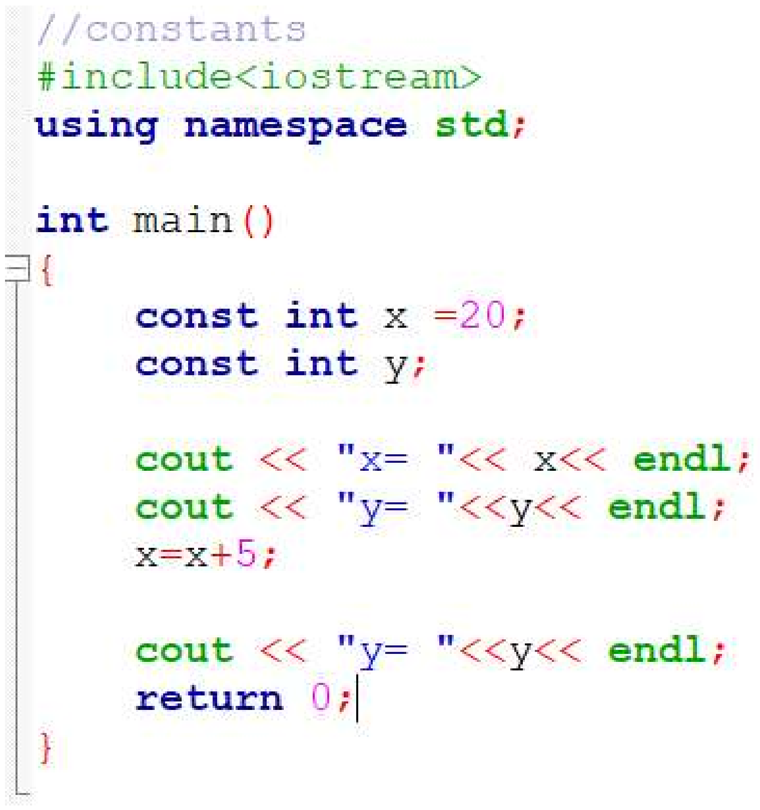
Declared Constant



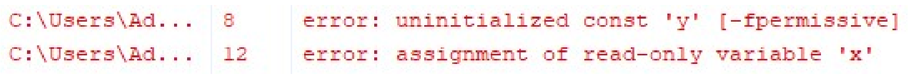
• Constants declared using the `const` keyword

```
const double pi {3.1415926};

const int months_in_year {12};

pi = 2.5;  // Compiler error
```

Example



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

int main()
{
    const int x =20;
    const int y;

    cout << "x= "<< x<< endl;
    cout << "y= "<<y<< endl;
    x=x+5;

    cout << "y= "<<y<< endl;
    return 0;
}
```



```
C:\Users\Ad...  8    error: uninitialized const 'y' [-fpermissive]
C:\Users\Ad...  12   error: assignment of read-only variable 'x'
```

- Error due do uninitialized constant y.
- Error assigning x a new value (x+5)

Character Literal Constant

- Special characters that are difficult or impossible to express otherwise in the source code of a program, like newline (\n) or tab (\t).
- All of them are preceded by a backslash (\).
- A list of some of such escape codes:

\n	newline
\t	tab
\v	vertical tab
\b	backspace
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\\	backslash (\)

Defined Constant

```
# define pi=3.142
```

Don’t use defined constants in Modern C++

Manipulator

- **Manipulators are operators used in C++ for formatting output.** The data is manipulated by the programmer's choice of display.

- There are numerous manipulators available in C++. Some of the more commonly used manipulators are provided here below:

endl Manipulator:

- This manipulator has the same functionality as the ‘\n’ newline character.

- **For example:**
```
cout << “IIT Roorkee" << endl;
cout << “Uttarakhand";
```

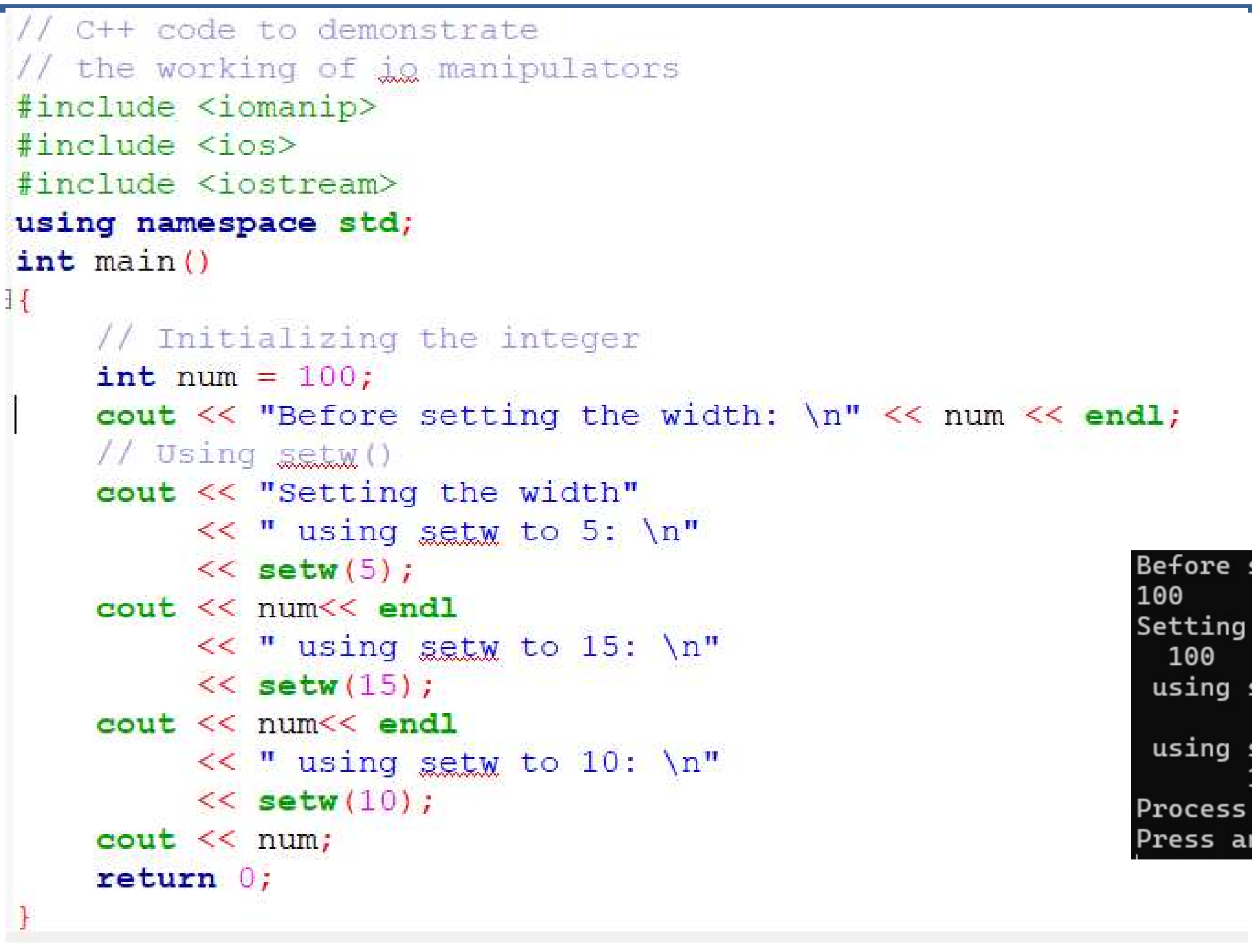
produces the output:

```
IIT Roorkee
Uttarakhand
```

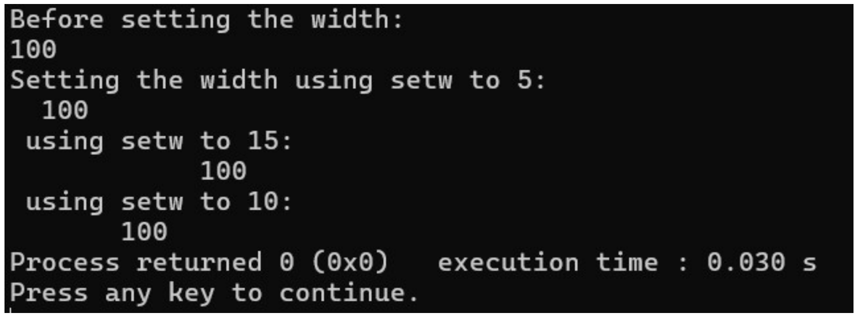
setw Manipulator:

- This manipulator sets the minimum field width on output.
- The syntax is: setw(x)
- Here, setw causes the number or string that follows it to be printed within a field of x characters wide, and x is the argument set in the setw manipulator (no effect if not followed by string or number).
- The header file that must be included while using setw manipulator is <iomanip>

Example



```cpp
// C++ code to demonstrate
// the working of io manipulators
#include <iomanip>
#include <ios>
#include <iostream>
using namespace std;
int main()
{
    // Initializing the integer
    int num = 100;
    cout << "Before setting the width: \n" << num << endl;
    // Using setw()
    cout << "Setting the width"
         << " using setw to 5: \n"
         << setw(5);
    cout << num<< endl
         << " using setw to 15: \n"
         << setw(15);
    cout << num<< endl
         << " using setw to 10: \n"
         << setw(10);
    cout << num;
    return 0;
}
```



```
Before setting the width:
100
Setting the width using setw to 5:
  100
 using setw to 15:
            100
 using setw to 10:
       100
Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

setfill Manipulator:

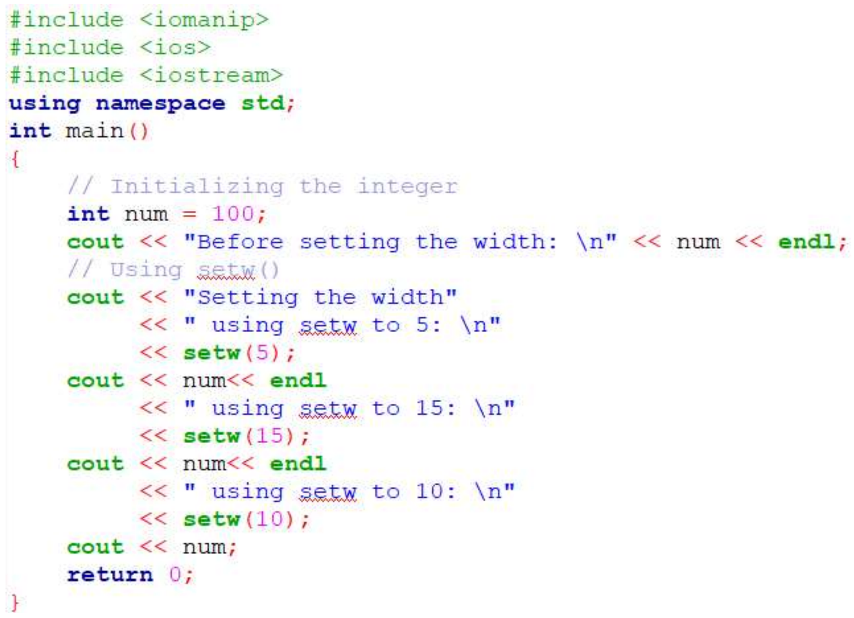
- This is used after setw manipulator. If a value does not entirely fill a field, then the character specified in the setfill argument of the manipulator is used for filling the fields.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main( )
{
cout << setw(10) << setfill(‘*') << 50 << “ “<<33 << endl;
}
```

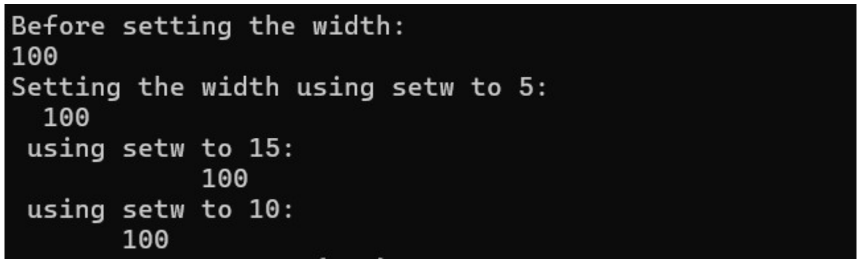
The output of the above program is

```
********50 33
```

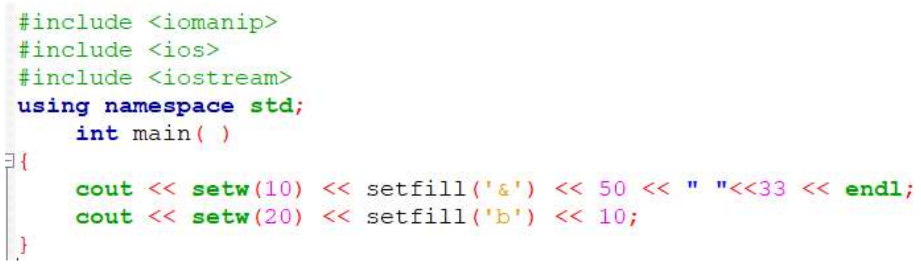
This is because the setw sets 10 width for the field and the number 50 has only 2 positions in it. So the remaining 8 positions are filled with * symbol which is specified in the setfill argument.



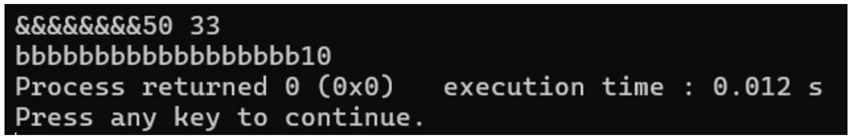
```cpp
#include <iomanip>
#include <ios>
#include <iostream>
using namespace std;
int main()
{
    // Initializing the integer
    int num = 100;
    cout << "Before setting the width: \n" << num << endl;
    // Using setw()
    cout << "Setting the width"
         << " using setw to 5: \n"
         << setw(5);
    cout << num<< endl
         << " using setw to 15: \n"
         << setw(15);
    cout << num<< endl
         << " using setw to 10: \n"
         << setw(10);
    cout << num;
    return 0;
}
```



```
Before setting the width:
100
Setting the width using setw to 5:
  100
 using setw to 15:
            100
 using setw to 10:
       100
```



```cpp
#include <iomanip>
#include <ios>
#include <iostream>
using namespace std;
    int main( )
{
    cout << setw(10) << setfill('&') << 50 << " "<<33 << endl;
    cout << setw(20) << setfill('b') << 10;
}
```

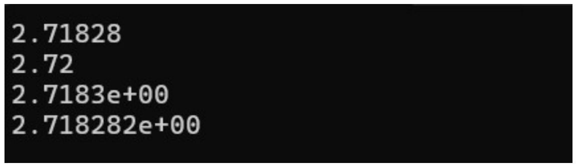


```
&&&&&&&&50 33
bbbbbbbbbbbbbbbbbb10
Process returned 0 (0x0)   execution time : 0.012 s
Press any key to continue.
```

setprecision Manipulator:

- The setprecision Manipulator is used with floating point numbers. It is used to set the number of digits printed to the right of the decimal point.
- This may be used in two forms:
  - **fixed**
  - **scientific**
- These two forms are used when the keywords **fixed** or **scientifi**c are appropriately used before the **setprecision** manipulator.
- The keyword **fixed** before the **setprecision** manipulator prints the floating point number in fixed notation.
- The keyword **scientific** before the **setprecision** manipulator prints the floating point number in scientific notation.

```cpp
  #include <iostream>
  #include <iomanip>
  using namespace std;
  int main( )
  {
      double x = 2.718281828459045;
      cout << setprecision(6)<< x << endl;
      cout << fixed << setprecision(2) << x << endl;
      cout << scientific << setprecision(4)<< x << endl;
      cout << setprecision(6)<< x << endl;
  return 0;
}
```



```
2.71828
2.72
2.7183e+00
2.718282e+00
```

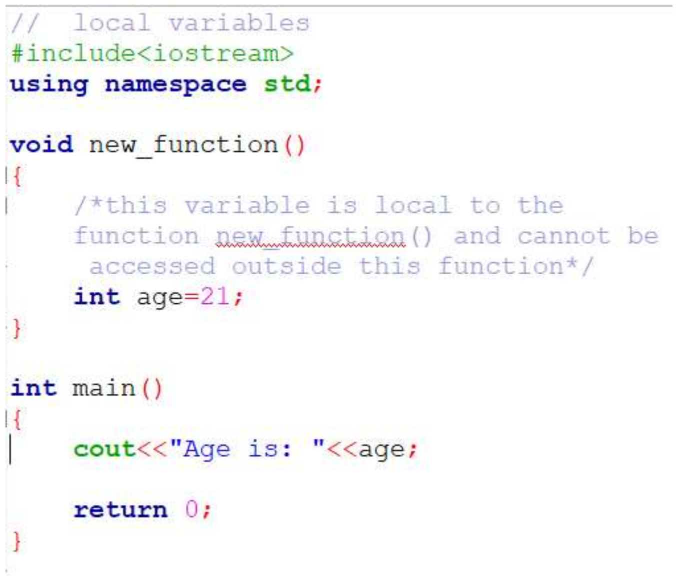
Scope of variable

- In general, the scope is defined as the extent up to which something can be worked with.
- In programming also the scope of a variable is defined as the extent of the program code within which the variable can be accessed or declared or worked with.
- There are mainly two types of variable scopes:
  - Local Variables
  - Global Variables

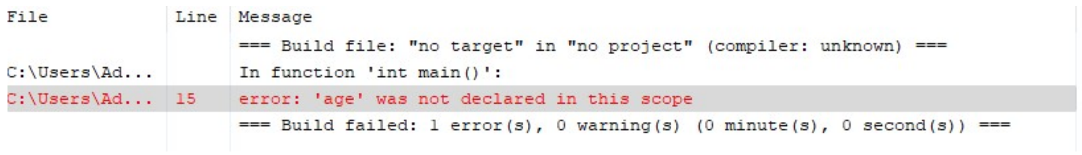
Local Variables

- Variables defined within a function or block are said to be local to those functions.
-
- Anything between ‘{‘ and ‘}’ is said to inside a block.

- Local variables do not exist outside the block in which they are declared, i.e. they **can not** be accessed or used outside that block.



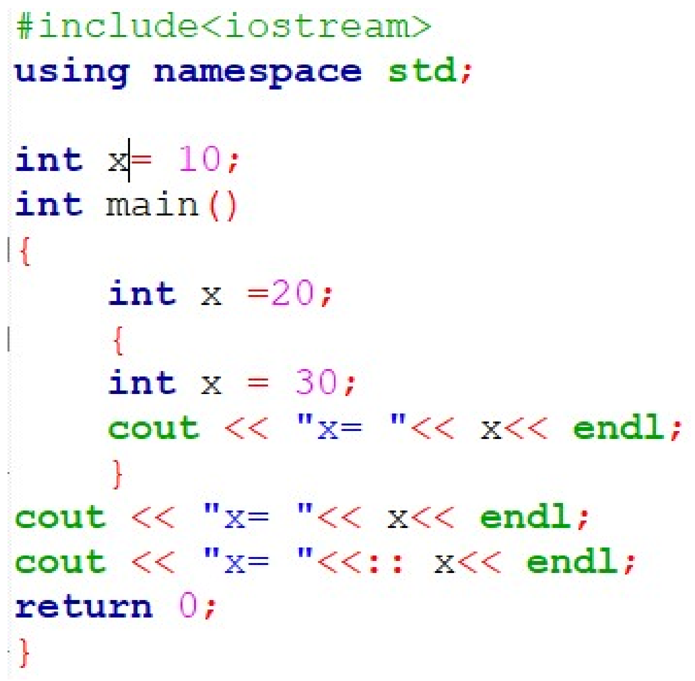
```
//  local variables
#include<iostream>
using namespace std;

void new_function()
{
    /*this variable is local to the
    function new_function() and cannot be
     accessed outside this function*/
    int age=21;
}

int main()
{
    cout<<"Age is: "<<age;

    return 0;
}
```



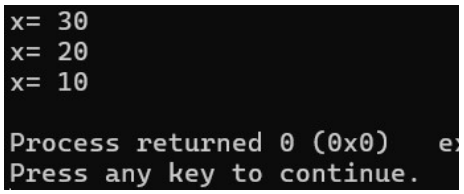
```
File              Line  Message
                        === Build file: "no target" in "no project" (compiler: unknown) ===
C:\Users\Ad...          In function 'int main()':
C:\Users\Ad...    15    error: 'age' was not declared in this scope
                        === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

Global Variable

- As the name suggests, Global Variables can be accessed from any part of the program.
- They are available throughout the lifetime of a program.
- They are declared at the top of the program outside all of the functions or blocks.
- **Declaring global variables**: Global variables are usually declared outside of all of the functions and blocks, at the top of the program. They can be accessed from any portion of the program.



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

int x= 10;
int main()
{
    int x =20;
    {
    int x = 30;
    cout << "x= "<< x<< endl;
    }
cout << "x= "<< x<< endl;
cout << "x= "<<:: x<< endl;
return 0;
}
```



```
x= 30
x= 20
x= 10

Process returned 0 (0x0)   e
Press any key to continue.
```