Data Type Modifiers

In this lesson, you will be introduced to data type modifiers.

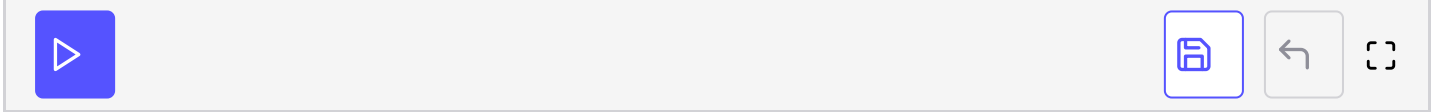
We'll cover the following

Introduction

The maximum value that can be stored in a variable of type `int` is `2147483647`. What if we want to store a value greater than `2147483647`?

Run the code below and see the output!

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

int main() {
  // Initialize variable
  int number = 2147483649;
  // Display variable value
  cout << number;

}
```



The maximum value of int

If we run the code given above, it does not give us the expected output. The above code should print `2147483649`, but it is printing `-2147483647` in output. So how can we handle values greater than the range of a data type? Similarly, how can we decrease the amount of space allocated to a particular variable? Here, data type

modifiers come to the rescue!

> ***Data type modifiers*** *are used with primitive data types to change the meaning of the predefined data types according to the situation.*

Data type modifiers in C++

We can use data type modifiers with `int` and `char` data types. C++ supports the following data type modifiers:

- long
- short
- unsigned
- signed

long

`long` is used to increase the length of a data type to `4` more bytes. We can use `long` with `int` and `double` data types. Let's use a `long` modifier with built-in data types.

```
#include <iostream>

using namespace std;

int main() {
  // Initialize variables
  int integer = 2147483649;
  long int long_integer = 2147483649;
  // Display variables value
  cout << "integer = " << integer << endl;
  cout << "long_integer = " << long_integer << endl;

}
```

long modifier

From the code above, we can see that we can precisely store values greater than the `int` range using a `long` modifier.

short

`short` decreases the available length of a data type to 2 bytes. We can use `short` with an `int` data type.

RUN the code below and see the output!

```
#include <iostream>

using namespace std;

int main() {
  // Initialize variables
  int integer = 32768;
  short int short_integer = 32768;
  // Display variables value
  cout << "integer = " << integer << endl;
  cout << "short_integer = " << short_integer << endl;

}
```

short modifier

The program given above generates unexpected results because `short int` reserves less space in memory.

An `int` reserves 4 bytes in memory. However, using a `short` modifier with `int` reserves 2 bytes in memory. Therefore, the maximum value that can be represented with `short int` is `32767`.

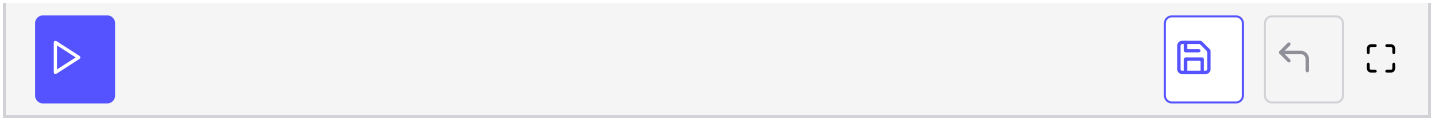
unsigned

`unsigned` allows us to store positive values only. We can use `unsigned` with `char` and `int` data types. With `unsigned int`, we can store any value from **0 to 4,294,967,295**. With `unsigned char`, we can store any value from **0 to 255**.

Run the program below!

```
#include <iostream>

using namespace std;

int main() {
  // Initialize variables
  int integer = -10;
  unsigned int unsigned_integer = -10;
```

```
  char character = 'A';
  unsigned char unsigned_character = 'B';

  // Display variables value
  cout << "integer = " << integer << endl;
  cout << "unsigned_integer = " << unsigned_integer << endl;

  cout << "character = " << character << endl;
  cout << "unsigned_character = " << unsigned_character << endl;

}
```



unsigned modifier

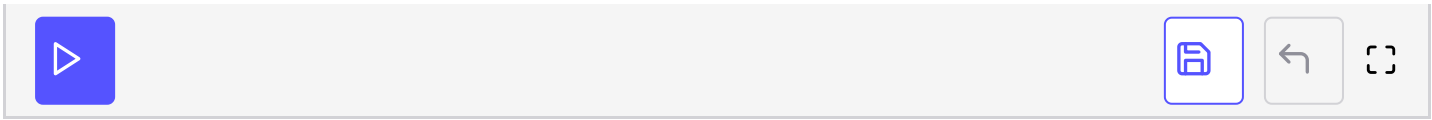
From the program above, it is clear that we cannot represent signed values with an `unsigned` modifier.

signed

`signed` allows us to store both positive and negative values. We can use `signed` with `char` and `int` data types.With `signed int`, we can store any value from **-2,147,483,648 to 2,147,483,647**. With `signed char`, we can store any value from **-128 to 127**.

Run the program below!

```
#include <iostream>

using namespace std;

int main() {
  // Initialize variables
  int integer = -90;
  signed int signed_integer = -90;

  char character = 'A';
  signed char signed_character = 'A';
  // Display variables value
  cout << "integer = " << integer << endl;
  cout << "signed_integer = " << signed_integer << endl;

  cout << "character = " << character << endl;
  cout << "signed_character = " << signed_character << endl;

}
```



signed modifier

From the program above, it is clear that we can use a `signed` modifier to represent signed values.

The table given below summarizes the data type modifiers.

Data Type	Keyword	Size in bytes	Values range
Integer	int	4	-2,147,483,648 to 2,147,483,647
Integer	short int	2	-32,768 to 32,767
Integer	long int	8	2,305,843,009,213,693,952 to 2,305,843,009,213,693,951
Integer	signed int	4	-2,147,483,648 to 2,147,483,647
Integer	unsigned int	4	0 to 4,294,967,295
Floating-point	float	4	+/- 3.4e +/- 38 (~7 digits)
Double	double	8	+/- 1.7e +/- 308 (~15 digits)
Long Double	long double	16	+/- 1.7e +/- 308 (~15 digits)
Character	char	1	-128 to 127
Signed Character	signed char	1	-128 to 127
Unsigned Character	unsigned char	1	0 to 255
Boolean	bool	1	0 or 1

Data type modifiers

> 📝 **Note:** From the above table, it is obvious that signed is the default declaration for `int` and `char` (`signed int` is similar to `int` and `signed char` is similar to `char`).

Q

```
int main() {
  unsigned float number = 18.9;
  cout << "Number = " << number << endl;
}
```

What is the output of the code given above?

Retake Quiz

Let’s discuss data type conversion in the next lesson.

Stay tuned!