Programming Fundamentals

By/ Merhan Atef

Section (2) Outline

1. Practice
2. Data Type
3. Modifying the Basic Types
4. Identifier Names
5. Variables
6. The Assignment Operator

Practice

- **Scenario:**
- **Program Name**: Simple Calculator
- **Description**: The program defines a function calculator which performs four basic arithmetic operations (addition, subtraction, division, and multiplication) on two integer values. This function is then called three times in the main function with different sets of values to demonstrate the results of these operations.

Practice (solution)

```
#include<iostream>
using namespace std;
void calculator(int x, int y)
{
cout << "sum of: " << x << "& " << y << " = "
<< x + y<<endl;
cout << "sub of: " << x << "& " << y << " = "
<< x - y << endl;
cout << "div of: " << x << "& " << y << " = "
<< x / y << endl;
cout << "mul of: " << x << "& " << y << " = "
<< x * y << endl;
}
```

```
int main()
{
calculator(8, 4);
calculator(12, 3);
calculator(10, 2);
system("Pause");
return 0;
}
```

The Six Basic Data Types

1. **Char** Stores a single character/letter/number, or ASCII values
   Ex :
   ```
   char myGrade = 'B';
   cout << myGrade;
   ```
2. **Int** → Stores whole numbers, without decimals. **Ex :** `int myNum = 5;`
3. **Float** → Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits. **Ex :** `float myFloatNum = 5.99;` (4 bytes)
4. **Double** → Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits. **Ex :** `double myDoubleNum = 9.98;` (8 bytes)
5. **String** → is used to store a sequence of characters (text). String values must be surrounded by double quotes. **Ex :** `string myText = "Hello";`
6. **Boolean** → Stores true or false values. **Ex :** `bool myBoolean = true;`

Modifying the Basic Types

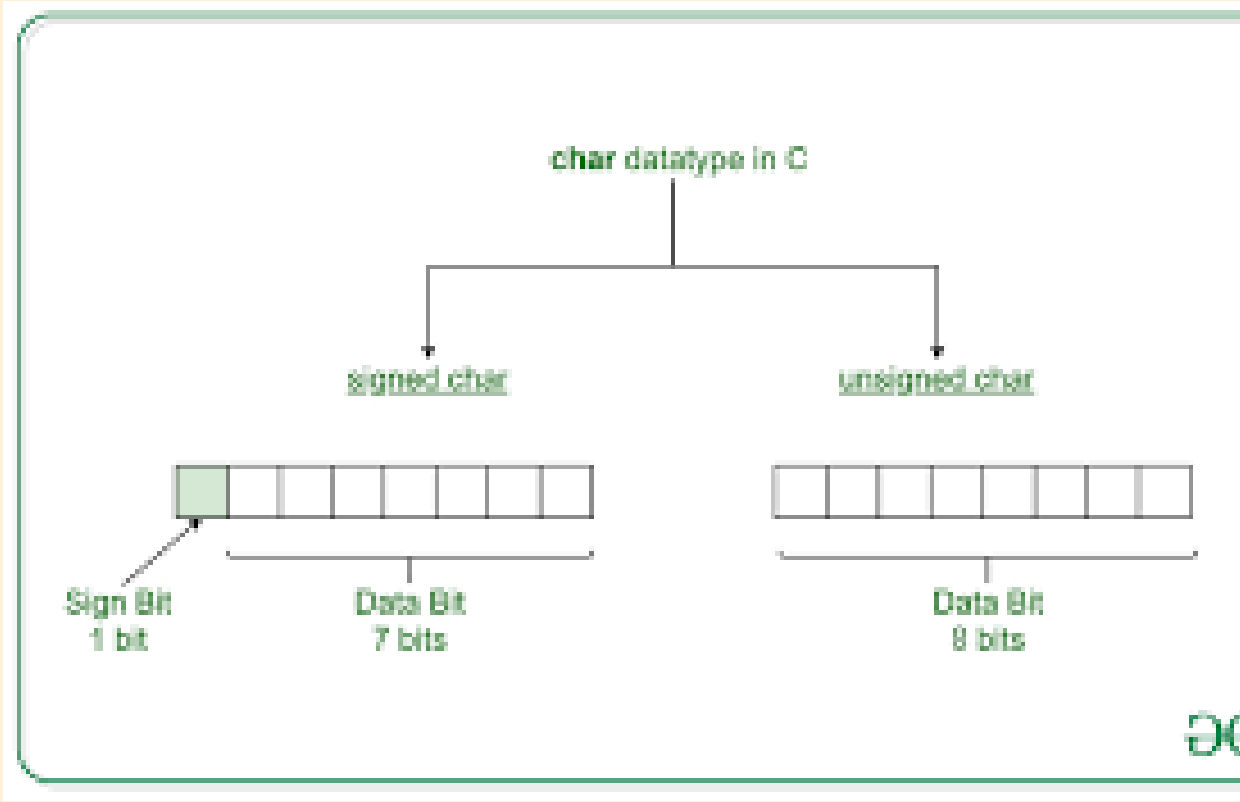
- Except for type **void**, the basic data types may have various modifiers preceding them.
  You use a modifier to alter the meaning of the base type.

- **The list of modifiers is shown here:**

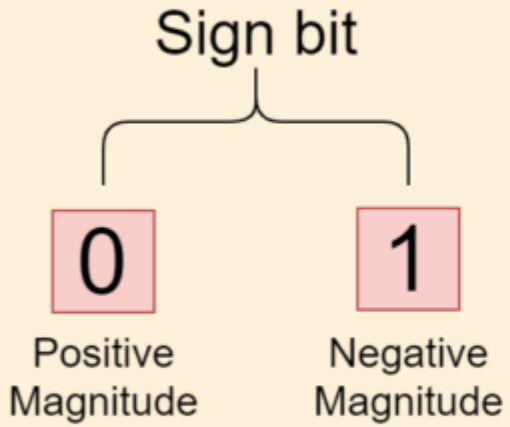
- Signed : It is used to specify that the variable can contain both positive and negative values.
- unsigned: It is used to specify that the variable can only contain positive values.
- long: It is used to specify that the variable will be of integer type but with a larger size.
- short : It is used to specify that the variable will be of integer type but with a smaller size.
- long long : It is used to obtain an integer with a size larger than the long type.

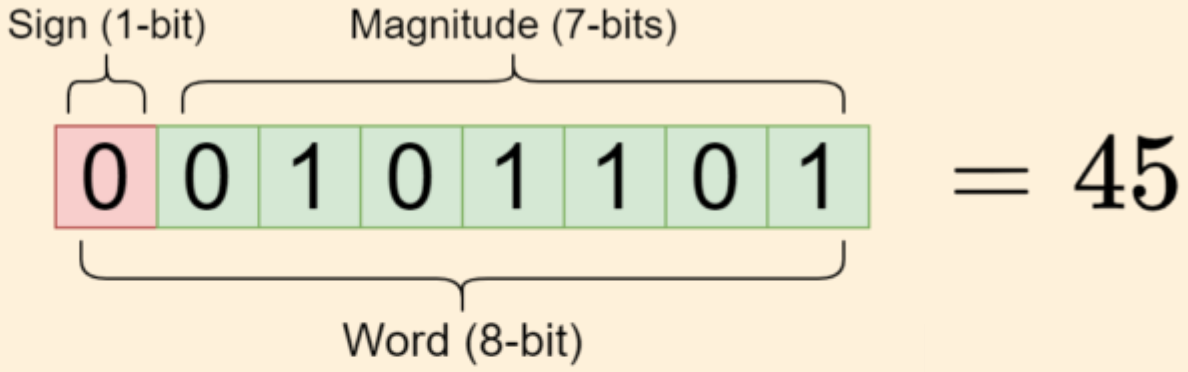
Modifying the Basic Types

- The difference between signed and unsigned integers is in the way that the high order bit of the integer is interpreted. If you specify a signed integer, the compiler generates code that assumes that the high-order bit of an integer is to be used as a *sign flag*. If the sign flag is 0, the number is positive; if it is 1, the number is negative.

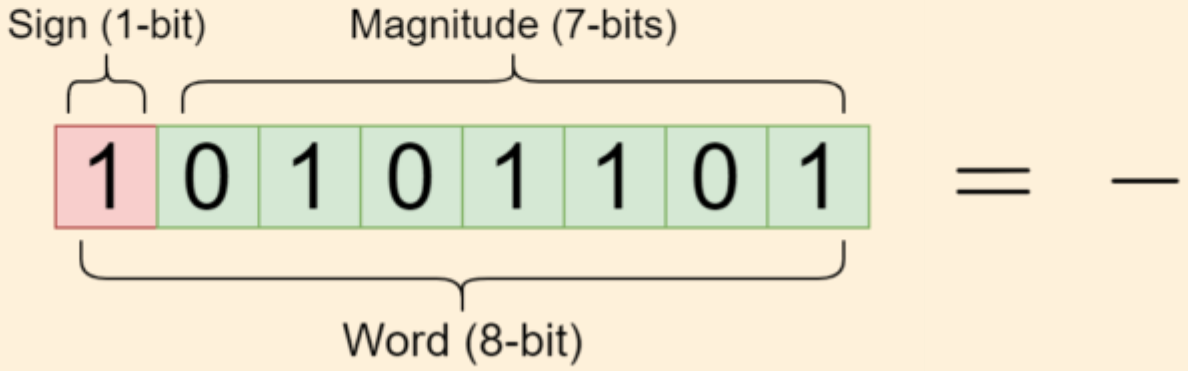


Modifying (signed)





$= 45$



$= -45$

Modifying (unsigned)

Signed value

Unsigned value

Modifying the Basic Types

- The size of both unsigned and signed numbers is set to n bits.
  Size = 2^n
- The range of numbers for an eight (8) bit unsigned number is from **"0" to "255"** .
  Range = 0 to $(2^n$ -1)

- Whereas, the signed binary number ranges from **"–127" to "127"** including the middle value of **"0"**.
  Range = - 2^{n-1} to $(2^{n-1}$ -1)

Modifying the Basic Types

- You can apply the modifiers **signed**, **short**, **long**, and **unsigned** to **integer**.
- You can apply **unsigned** and **signed** to **characters**.
- You may also apply **long** to **double**.

- The use of **signed** on integers is allowed, but redundant because the default integer declaration assumes a signed number. The most important use of **signed** is to modify **char** in implementations in which **char** is unsigned by default.

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

Identifier Names

- In C/C++, the names of variables, functions, labels, and various other user-defined objects are called identifiers.
- **Rules to Name of an Identifier in C++:**

1. An identifier can consist of letters (A-Z or a-z), digits (0-9), and underscores (_). Special characters and spaces are not allowed.

2. An identifier can only begin with a letter or an underscore only.

3. C++ has reserved keywords that cannot be used as identifiers since they have predefined meanings in the language. For example, int cannot be used as an identifier as it already has some predefined meaning in C++. Attempting to use these as identifiers will result in a compilation error.

4. Identifier must be unique in its namespace.

Identifier Names

- In an identifier, upper- and lowercase are treated as distinct. Hence, **count**, **Count, and COUNT** are three separate identifiers.
- An identifier cannot be the same as a C or C++ keyword, and should not have the same name as functions that are in the C or C++ library.

Correct	Incorrect
Count	1count
test23	hi!there
high_balance	high...balance

Variables

- **variable** is a named location in memory that is used to hold a value that may be modified by the program.
- All variables must be declared before they can be used.

The general form of a declaration is:

Data type variable_list;

Here are some declarations:

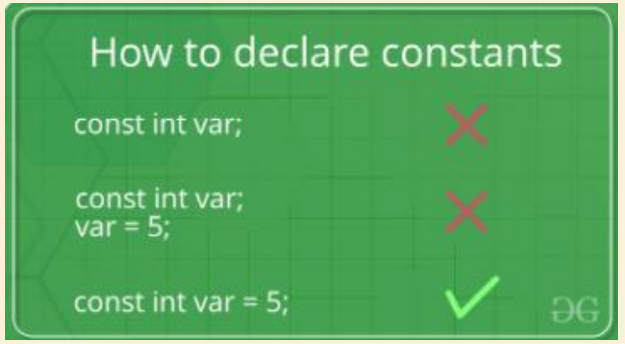
```
int i,j,l;
short int si;
unsigned int ui;
```

Const variables

- Variables of type const may not be changed by your program. (A const variable can be given an initial value, however.)

const int a=10;

- **Constant Variables:**
- There are certain set of rules for the declaration and initialization of the constant variables:
- The const variable cannot be left un-initialized at the time of the declaration.
- It cannot be assigned value anywhere in the program.
- Explicit value needed to be provided to the constant variable at the time of declaration of the constant variable.



Variable Initializations

- You can give variables a value as you declare them by placing an equal sign and a value after the variable name.
- **The general form of initialization is:**

type variable_name = value;

- Examples

```
char ch = 'a';
int first = 0;
float balance = 123.23;
```

Cast

- The **casting operators** is the modern C++ solution for converting one type of data safely to another type.
- You can force an expression to be of a specific type by using a cast. The general form of a cast is:

(type_name) expression

```
#include<iostream>
using namespace std;
void main()
{
int x = 5;
cout<<(float)x/2<<endl;
system("Pause");
}
```

```
#include<iostream>
using namespace std;
void main()
{
int x = 5;
cout<<x/2<<endl;
system("Pause");
}
```

Cast

```
#include<iostream>
using namespace std;
void main()
{
cout << (int) 3.4<<endl;
cout << (double) (25) / 2 << endl;
cout << (double)(25 / 2) << endl;
cout << (int) (3.8+ (double(13)/2)) << endl;
system("Pause");
}
```

The Assignment Operator

- **The general form of the assignment operator is:**

variable_name = expression;

Example 1:

```
#include<iostream>
using namespace std;
void main()
{
int x;
x = 5;
cout << "x = " << x<<endl;
system("Pause");
}
```

Example 2:

```
#include<iostream>
using namespace std;
void main()
{
int x,y,z;
y=z=5;  // Multiple
Assignments
x = y + z;
cout << "x = " << x<<endl;
system("Pause");
}
```

Type Conversion in Assignments

- The value of the right side (expression side) of the assignment is converted to the type of the left side (target variable).
- **Example:**

```
#include<iostream>
using namespace std;
void main()
{
int x = 65;
char ch = 'a';
float f = 1.4;
ch = x;
x = f;
f = ch;
cout << x <<endl<< ch<<endl << f<<endl;
system("Pause");
}
```

Example

1. The following program contains several errors:

```
#include <iostream>

using namespace std;

void main()
{
    char first = 'ahmed';
    const int x = 4;
    long void y ;
    cout >> "Programming Fundamental Course " ;
    cout << endl cout << 'Study hard please ';
    cout < endl;
    x= 88;
    cout << "x=" << x << endl;
    return 0;
}
```

Do you have any questions ?

Thank you

piece of advice

Aim for the impossible and don’t stop until you’ve made it possible