

Variables, variables types and the console in C++

During our first practice we talked about a few things. Here we will list and expand on some of them. First, let's start with a brief explanation about the relation between our code and the console. C++ provides a **namespace** which is called **std**, std stands for **C++ Standard Library**. A namespace is used to organize the code, for example you can have two functions with the same name in two different namespaces just as you can have two files with the same names in two different folders, you can safely use these two files, as long as you specify which folder they belong to, the same thing applies to functionality from namespaces.

The C++ Standard Library provides functionality which let us "communicate" with the console. As our program is being executed, sometimes there are instructions which use the console, in most cases this is because of one of two reasons, the first being that we want to take some **input** from the console and second being that we probably want to **output** something to the console. In other words, we may want to either **read** from the console or **write** to the console.

Let's take a look at examples for **input** and then for **output**:

Output:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "1. Output - Writing to the console." << endl;

    return 0;
}
```

- **cout** writes everything after the << on the console. Of course, there are variations to this which are shown in the code below.
- **endl** stands for "end line", which moves the cursor of the console to the beginning of the next line.
- We will refer to everything in-between double quotes - "text in-between double quotes" as **text**, more information is given later. For now, we need to know that whatever we put in double quotes after **cout <<** will be written(printed) to the console.

If we want to output(write) the **value** of a **variable** to the console, we can do it like this:

```
#include <iostream>


using namespace std;

int main()
{
    int numberOfMartinsAndVickies = 3;
    cout << "There are " << numberOfMartinsAndVickies << " Marties and "
         << numberOfMartinsAndVickies << " Vickies in our group." << endl;

    return 0;
}
```

The compiler literally takes the value of **numberOfMartinsAndVickies**, which is **3** and slaps it everywhere it sees **numberOfMartinsAndVickies** below the line which initialized it.

The output of this program is:

 Microsoft Visual Studio Debug Console

```
There are 3 Marties and 3 Vickies in our group.
```

Input

```
#include <iostream>

using namespace std;

int main()
{
    int age;
    cin >> age;

    return 0;
}
```

Reading from the console naturally requires us to save the information we are getting from the console in our memory so we can use it later in our code. We can read and save multiple variables by **chaining** the >> operator.

- When we talk about **memory**, we refer to RAM – random access memory, this is the memory which our program uses when it is running.
- **Chaining the >> operator** refers to using the >> **operator** multiple times with one **cin** as in the example below. In the example for output we used **chaining** with the << **operator** and **cout** to put values from variables in-between text we wanted to write to the console.
- **Operator** is just some that takes arguments and does something with them, just as the **+** **operator** return the summation of two numbers, the >> **operator** puts values that are read from the console into the variables specified after it – for example, the variables *whole* and *number* below.

This code reads two numbers from the console, one whole number and one real number and writes a message to the console with the values of the variables:

```
#include <iostream>

using namespace std;

int main()
{
    // Declare variables.
    int whole;
    double real;

    // Ask the user for input. (Write to the console)
    cout << "Please enter one whole and one real number." << endl;

    // Read the two numbers from the console. (Read from the console)
    cin >> whole >> real;

    // Write relevant message with the information entered from the console.
    cout << "The numbers you entered are: " << whole << " " << real << endl;

    return 0;
}
```

This explanation naturally leads to a need for a deeper understanding of variables and their types. Here we will show some of the primitive(built-in) types in the C++ language. Opposed to **built-in** types are the **user defined** data types, which obviously, we define. We will talk about them in the future.

As you've seen above this is how we create a variable of type **int**:

```
int number = 42;
int variableWithValue;
```

- **int** is the type of the variable.
- **number** is the name of the variable.
- **42** is the value of this variable.

Using the logic above and knowing that we can create different types of variable with different names and values, we can conclude that the generic form for variable creation is such:

<type> <name> = <value>;

Or

<type> <name>;

There are different types which we will explain shortly, but first let's take a look at **variableWithNoValue**, this variable does not have a value which was explicitly given by us (we can contrast this with the variable **number**, with which we used the **assignment operator** "=", to give the value **42**. This means that it was given a block of memory, but no value was written in that memory. This means that the value of this variable can be any whole number (i.e. any value of type **int**). In most cases the compiler won't let you use an uninitialized variable because it leads to **unexpected behavior** of the program.

Variable Types

- **int** – Can hold any whole number.
Examples: 0, 1, 2, 999999, -1, -3, -555343...
- **char** – Can hold a single character, you can think of character as letter of an alphabet, but here the alphabet does not only contain letter, but also numbers and special symbols. We show that something is of type **char** by surrounding it with single quotes – '<character>'.
Examples: 'a', 'b', '1', '[', '\$', '!', ...
- **double** – Can hold real numbers, precision after the dot is around 16 symbols, that means when you make calculations by hand and using the computer, if you are correct, the first 16 digits after the dot of your answer and the answer of the computer should be the same, the computer starts to lose precision after the 16th digit because memory is limited 😞 (We will elaborate on that in the second practice).
Examples: 3.14, 3.00, -13.33, 420.420
- **float** – Basically the same as **double**, but the precision is 7-8 digits after the dot. It takes less space though.
- **bool** – This type has only two values – *true* and *false*. It is the result of expressions like **x > 6**, we can use the value of such expression with some C++ constructions as the **if-statement** as we will see in the future. For now, we need to remember that it takes two values – *true* and *false*.
Examples: bool isLegal = false; bool prison = true;
- ****text** – There is no such type in C++ but for now in order to not confuse ourselves we will call things like **"This is text"** just text. The formal name for values like **"random text is used here"** (including the double quotes) is **string literal**.

Last but not least, in C++ there is something called **type conversion**, we will talk about it in the future but for now we will leave you with some questions regarding it and how C++ saves variables in memory. These are just small things you can try to compile & run and see if the answers surprise you.

- Question 1: What happens when you write a character to the console?
- Question 2: What happens when you write a bool to the console?
- Question 3: What happens when we try to save the character 'a' in an int variable? What is the value of the variable?
- Question 4: What happens when you save the sum of two variables of type double in a variable of type integer?
- Question 5: What happens when you try to assign a string literal to a variable of type int. What does that mean?

- Coding task 1: Write a program that reads a number from the console which represents the temperature in Celsius and output the temperature in Fahrenheit to the console. The formula for conversion from Celsius to Fahrenheit is **Fahrenheit = (Celsius * 9 / 5) + 32**. Check your answers using online Celsius to Fahrenheit calculator. If you have any questions, message us.

- Coding task 2: Do the same thing, but instead of converting from Celsius to Fahrenheit, convert from Fahrenheit to Celsius. Calculate the formula for Celsius by yourself. Check your answers using the online calculators and message us if any questions arise.