Lab A

31 January 2025 07:14

Q1. Hello World

Locate the Solution Explorer within Visual Studio and select the Hello World project. Right click on this project and select Build. This should compile and link the project. Now run the Hello World program.

Change between Debug and Release mode. Compile again and rerun the program.

Solution

N/A

Test Data

N/A

Sample Output

N/A

Reflection

After building the project, I compiled the program in both build and release mode. I did not see any change in compile time or any other changes.

Further Information

What is the difference between both modes? What do they do?

Q2. Creating a New Project

Create a new Empty C++ Console project called Temperature by using the project application wizard.
Write a program to input a Fahrenheit measurement, convert it and output a Celsius value.

Solution

```
int main(int argc, char** argv) {
    cout << "Enter a temperature in Fahrenheit:\n";
    int fahrenheit;
    cin >> fahrenheit;
    float celsius = 5.0 / 9.0 * (fahrenheit - 32);
    cout << "The temperature in Celsius is " << celsius << endl;
}
```

Test Data

N/A

Sample Output

```
Enter a temperature in Fahrenheit:
788
The temperature in Celsius is 420
```

Reflection

When using int types for Celsius and Fahrenheit, the result is truncated. Since 5/9 is 0.555…, the result is always 0. By using 5.0/9.0 we use floating point division, meaning the result is not truncated. Using a float for Celsius also gives us a more accurate output, as an int would truncate the decimal part.

Further Information

Q3. Types

Write a program that prints out the size in bytes of each of the fundamental data types in C++.

Solution

```
cout << "The size of a short is " << sizeof(short) << " bytes.\n";
cout << "The size of an unsigned short is " << sizeof(unsigned short) << " bytes.\n";
cout << "The size of an integer is " << sizeof(int) << " bytes.\n";
cout << "The size of an unsigned integer is " << sizeof(unsigned int) << " bytes.\n";
cout << "The size of a long is " << sizeof(long) << " bytes.\n";
cout << "The size of an unsigned long is " << sizeof(unsigned long) << " bytes.\n";
cout << "The size of a float is " << sizeof(float) << " bytes.\n";
cout << "The size of a double is " << sizeof(double) << " bytes.\n";
cout << "The size of a character is " << sizeof(char) << " bytes.\n";
cout << "The size of an unsigned character is " << sizeof(unsigned char) << " bytes.\n";
cout << "The size of a boolean is " << sizeof(bool) << " bytes.\n";
```

Test Data

N/A

Sample Output

```
The size of a short is 2 bytes.
The size of an unsigned short is 2 bytes.
The size of an integer is 4 bytes.
The size of an unsigned integer is 4 bytes.
The size of a long is 4 bytes.
The size of an unsigned long is 4 bytes.
The size of a float is 4 bytes.
The size of a double is 8 bytes.
The size of a character is 1 bytes.
The size of an unsigned character is 1 bytes.
The size of a boolean is 1 bytes.
```

Reflection

The sizeof() function returns the size of a data type in bytes. I can also be used to find the size of an array, or the total size of objects in a class.

Further Information

Q4. Floating Point Precision

Write a simple program that includes the lines:

```
double x = 10.0;
double y = 10.0;
if (x == y)
      cout << "X and Y are identical" << endl;
```

Now try y = 20.0 / 2.0 and execute the program again.
Then try a more complex calculation for y e.g.

```
const double x = 100000.123456789;
const double a = 200000.123456789;
double y = (x + a) / x;
double z = 1.0 + (a / x);
if (y == z)
   cout << "y and z are identical" << endl;
```

Once you're confident you understand the logic, investigate:

```
double z = x / y;
```

How small does y have to be before you get a "divide by zero" error? Does the value of x affect the result?

Solution

```
const double x = 100000.123456789;
const double a = 200000.123456789;
double y = (x + a) / x;
double z = x / y;

if (y == z) {
    cout << "Y and Z are identical" << endl;
} else {
    cout << "Y and Z are NOT identical" << endl;
}
```

The smallest positive normalized double value in C++ is **2.22507×10^{-308}.**
Any number smaller than this (but nonzero) is a denormalized number and leads to overflow.
If Y == 0, then the result is infinity
The larger the value of X, X/Y grows exponentially and may hit infinity sooner.

Test Data
N/A

Sample Output
N/A

Reflection
The first program outputs both X and Y as identical. This is because no operations are done to either variable, meaning no rounding errors are possible.
In the second program, although mathematically Z and Y are equal, we introduce some rounding errors. When adding x + a, there are likely rounding errors as they are large numbers. Then, when performing division ((x + a) / x), there are additional rounding errors. This leads to Y not being equal to X.

Further Information

Q5. C#/C++ Iteration Comparison

```
static void Main(string[] args)
{
   int factorialNumber = 5;
   int factorialTotal = 1;

   for(int n = 2; n <= factorialNumber; ++n)
   {
      factorialTotal *= n;
   }

   System.Console.WriteLine(factorialTotal);
}
```

Port the above C# code in to C++ using the provided main.cpp file.

Solution

```
int main(int argc, char* argv[])
{
    int factorialNumber = 10;
    int factorialTotal = 1;

    for (int n = 2; n <= factorialNumber; ++n) {
        factorialTotal *= n;
    }

    cout << factorialTotal << endl;
}
```

Test Data
N/A

Sample Output
```
3628800
```

Reflection
The syntax of the main function had to be changed, as this is different in C++
The for loop, and its contents, remained the same as the syntax is identical here.
The output to console syntax was changed.

Further Information

Q6. Calculate Average Using Iteration

Using a while loop (or do-while loop), calculate the average value of values provided by the user from the console (cin). You should calculate the average after the user either enters a negative number or the user enters a non-number value (e.g. a letter).

Solution

```
int main(int argc, char** argv) {
    int numProcessed = 0;
    double sum = 0;
    double value;

    while ((cin >> value) && (value > 0)) {
        sum += value;
        ++numProcessed;
    }
    cin.clear();

    if (numProcessed > 0) {
        double average = sum / numProcessed;
        cout << "Average: " << average << endl;
    }
    else {
        cout << "No list to average." << endl;
    }
}
```

Test Data

N/A

Sample Output

```
23
5
33
2
exit
Average: 15.75
```

Reflection

I learnt that putting (cin << value) inside the condition for a while loop will continue to take input as long as it is appropriate for that data type.
I also learnt that cin.clear() clears any error flags on the input stream, allowing the program to continue.

Further Information