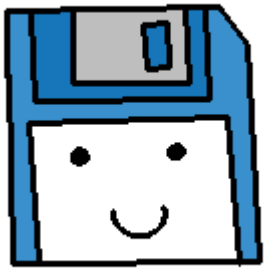
C++ Basics

Written by Rachel J. Morris, last updated 2015-01-14

An empty C++ program



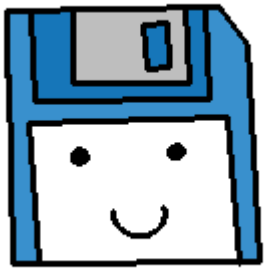
main.cpp ✖

```
1  int main()
2  {
3      return 0;
4  }
5
```

This is an empty C++ program – the bare minimum you need to get started.

The **`main()`** function is the starting point. Without it, you will get a compile error.

An empty C++ program



main.cpp ✖

```
1  int main()
2  {
3      return 0;
4  }
5
```

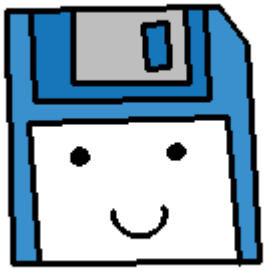
← Start

← End

The **`main()`** function is the starting point.
Without it, you will get a compile error.

The **`return 0;`** is where the program ends.
The 0 means that everything finished properly.

An empty C++ program



main.cpp

```
1  int main()
2  {
3      return 0;
4  }
5
```

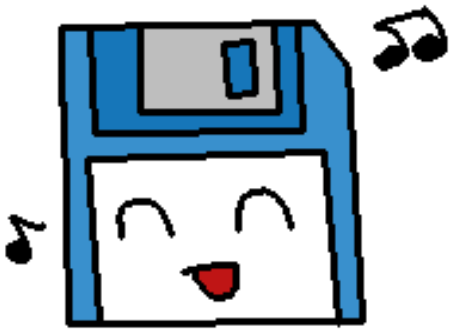
Start

End

The **`return 0;`** is a throwback to an old way of handling errors – 0 is no errors, and then returning *any other number* would stand for some error code.

Ever seen a program give you an error like
“Error code 201”
Which isn't helpful at all?

An empty C++ program



main.cpp ✖

```
1  int main()
2  {
3      return 0;
4  }
5
```

← Start

← End

This program will begin and end immediately.

We need to add more!

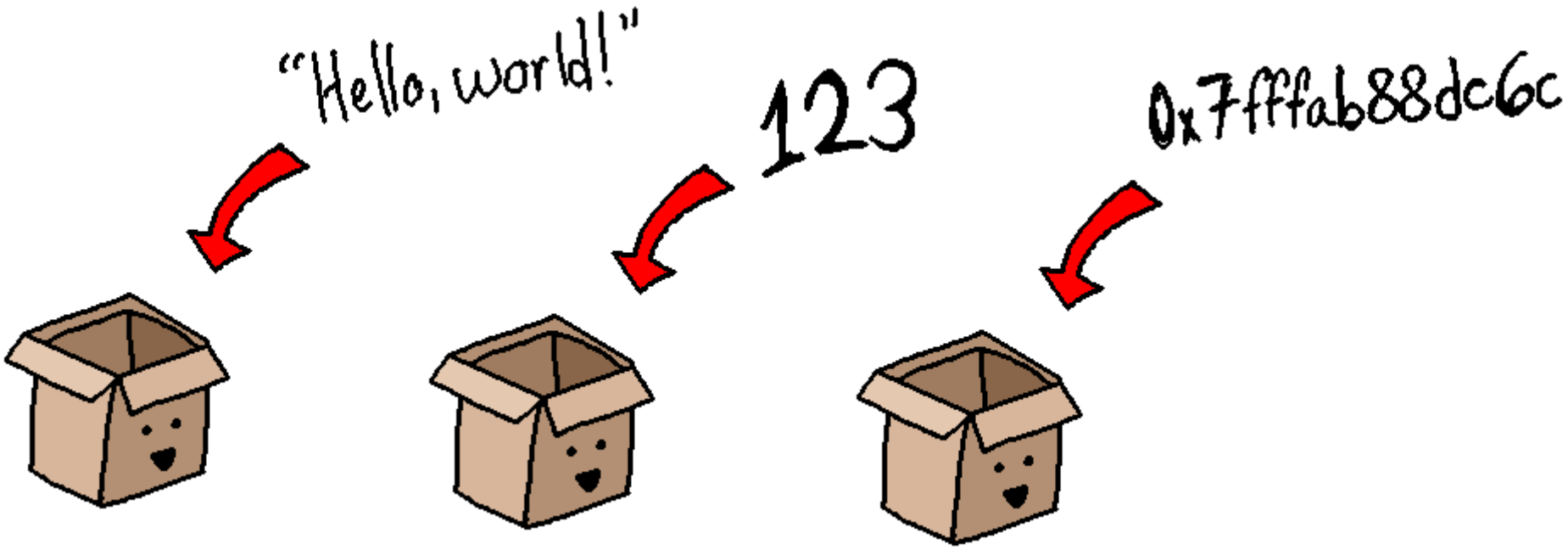
Variables

Written by Rachel J. Morris, last updated 2015-01-14

Variables

A **variable** is a place where you can store data.

Numbers, letters, text, memory addresses, etc.



Variables

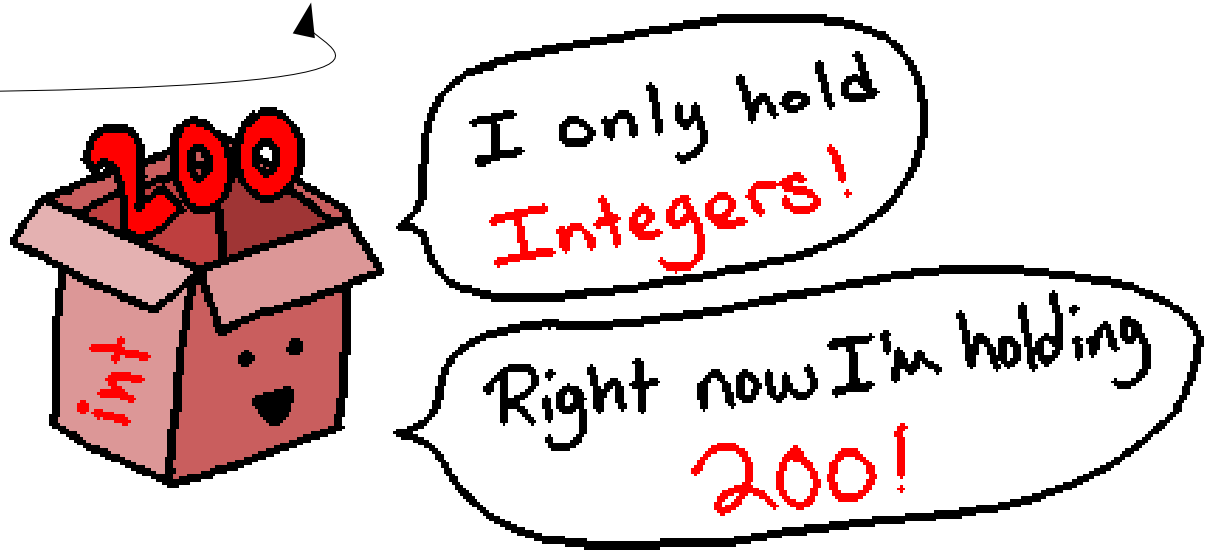
In C++, you must **declare** a variable before you can use it.

When you declare it, you must specify the **data type** and the **name (identifier)** for the variable.

Data Type

```
int age = 200;
```

Identifier



Variables

If you do not **initialize** the variable with a value, it is initially stored with **garbage**.

Garbage is the term for a value at some memory address. When we create a new variable, we reuse a memory address that isn't being used anymore.

When something stops using memory, it doesn't necessarily clean up its data; it's just easier to leave it.

Variables

```
1  int main()
2  {
3      int numA;
4      int numB;
5      int sum = numA + numB;
6
7      return 0;
8  }
9
```

Watches (new)

Function arguments	
Locals	
numA	32767
numB	0
sum	32767

In this program, we have **numA** and **numB**, but never assigned a value.

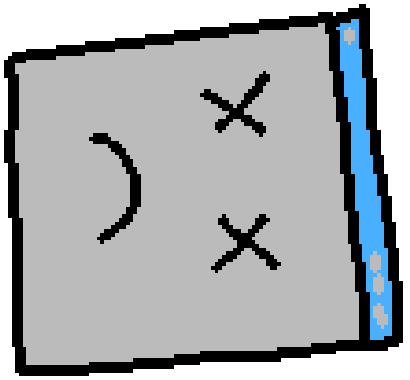
numB got the value 0, but **numA** pulled a value from that memory address, so it is 32767.

Variables

```
1  int main()
2  {
3      int numA;
4      int numB;
5      int sum = numA + numB;
6
7      return 0;
8  }
9
```

Watches (new)

Function arguments	
Locals	
numA	32767
numB	0
sum	32767



Having **garbage data** in your program (by not initializing variables) can create **undefined behavior**.

This means you can't predict what your program will do!

Variables

Here are a few ways we can declare a variable.

Declaring a variable, then assigning a value

```
int a;
a = 20;
```

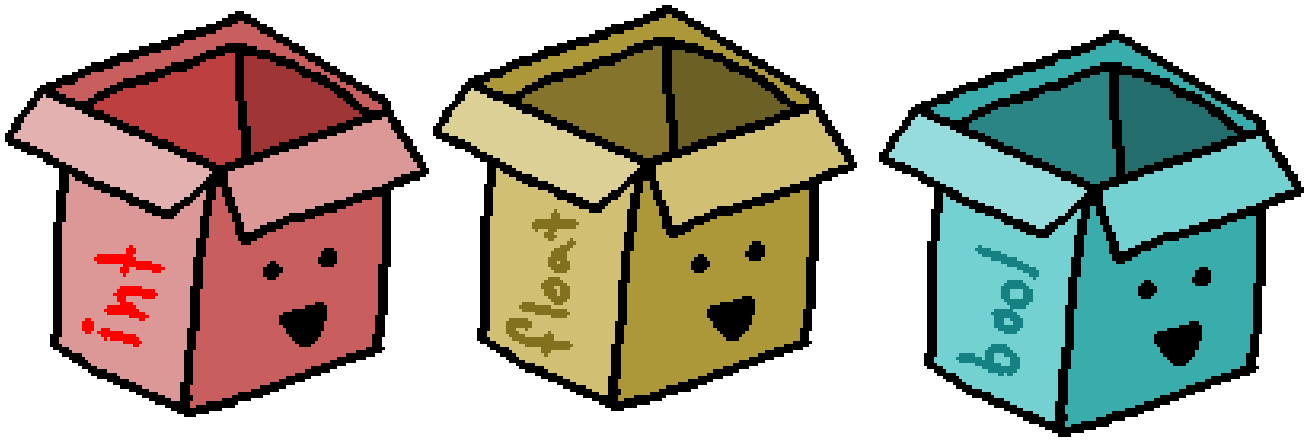
Assigning a value on declaration

```
int b = 20;
```

Declaring multiple integer variables at once

```
int x, y, z;
```

Data Types



Written by Rachel J. Morris, last updated 2015-01-14

Data Types

Primitives (built-in)

- Integer `int`
- Float `float`
- Boolean `bool`
- Character `char`

C++ Standard Library

- String `string`
- Input stream `istream`
- Output stream `ostream`
- vector, list, and more...

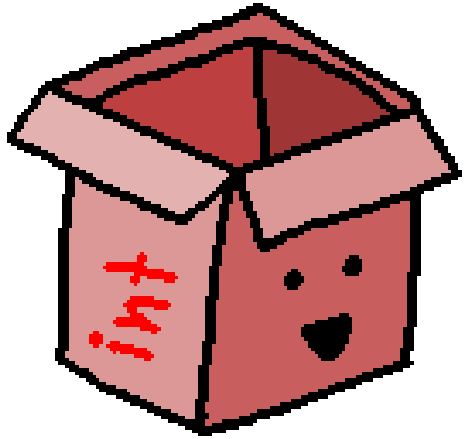
Data Types

Integer

```
int age = 22;
```

Stores whole numbers – no decimals

Positive, negative, or zero



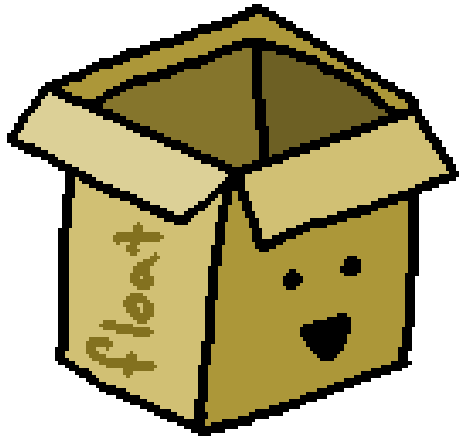
Data Types

Float

```
float money = 9.99;
```

Stores numbers with decimal (or without decimal).

Positive, negative, zero.

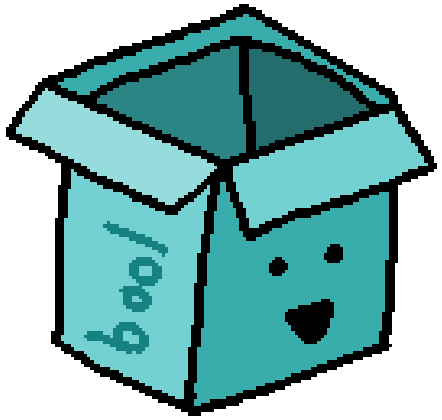


Data Types

Boolean

```
bool isProgrammer = true;
```

Stores “true” or “false”.

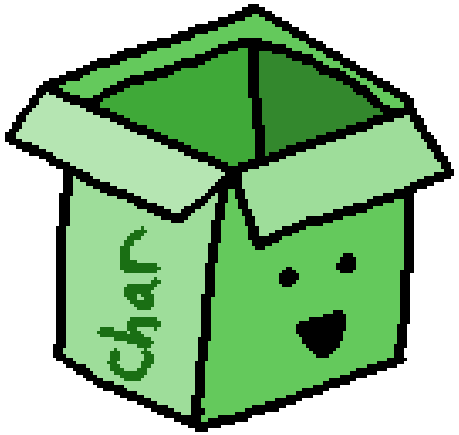


Data Types

Character

```
char currency = '$';
```

Stores any one character:
Letter, number, symbol, key-code, etc.

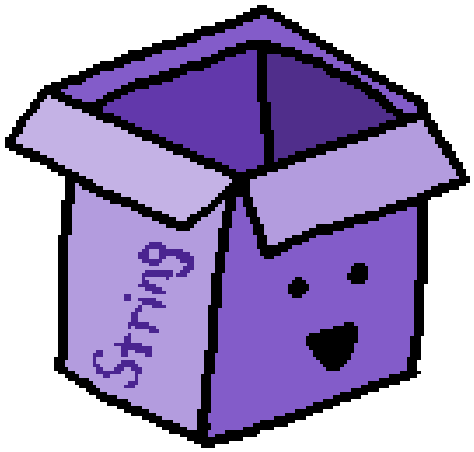


Data Types

String
(C++ Standard Library, requires the **`string`** library)

```
string firstName = "Elaine";
string lastName = "Threepwood";
string favoriteColor = "purple";
string explitive = "@^!%@#*!!";
```

Stores (virtually) any amount of text – Letters, numbers, symbols, etc.



Input / Output

Written by Rachel J. Morris, last updated 2015-01-14

Output

To use the **`cin`** (console-in) and **`cout`** (console-out) commands in C++, we need to first import a library...

```
#include <iostream>
```

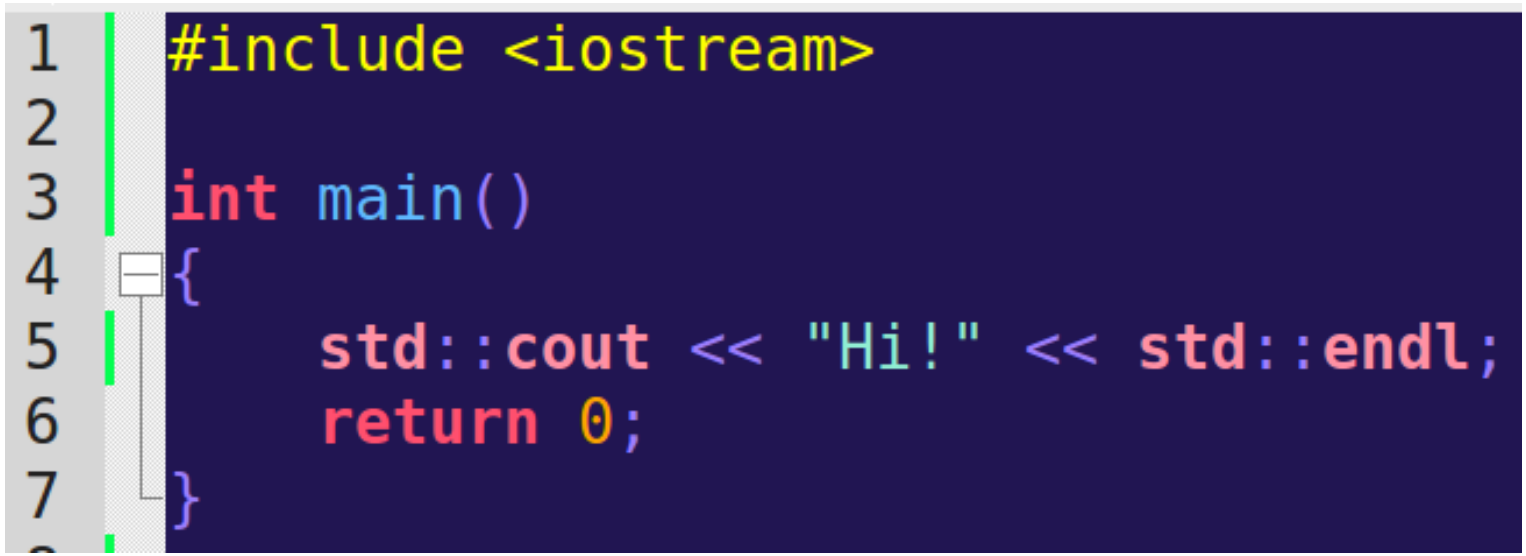
To use another library, we use
`#include <library_name>`

`iostream` is for INPUT-OUTPUT streaming.

Output

Include library

Output “Hi” and end line



```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hi!" << std::endl;
6      return 0;
7  }
```

The **`cout`** command is part of the **Standard Library**.
We prefix **`cout`** here with **`std::`** (standard) to specify the **namespace** it comes from.

`endl` stands for end-line. We have to manually specify where line-breaks are in C++.

Namespace?

```
namespace mymath
{
    int Sum( int a, int b )
    {
        return a + b;
    }
}
```

A **Namespace** is a way to group code together under a common name.

If you are using multiple libraries, it is possible that there could be **naming collisions** – two libraries using the same names for functions or variables!

The Standard Library has its functions in the **std namespace.**

Namespace?

```
std::cout << "One ";
std::cout << "Two ";
std::cout << "Three ";
```

We will learn more about namespaces later.

Just remember that functionality from **`iostream`** (And other libraries we will use) are in the **std** namespace.

Output

```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hi!" << std::endl;
6      return 0;
7  }
```

We can get rid of these **`std::`** prefixes by adding

`using namespace std;`

To the beginning of our program, outside of **`main()`**.

Output

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      cout << "Hi" << endl;
7
8      return 0;
9  }
```

We can get rid of these **`std::`** prefixes by adding

`using namespace std;`

To the beginning of our program, outside of **`main()`**.

Output

The **<<** signs after **`cout`** is the **Stream Operator**.

```
cout << "Hello!" << endl;
```

Remember that, for console-output, the stream operator points ***towards*** the output.

We can keep linking **string literals**, variables, and **`endl`** end-lines together with the stream operator.

```
int year = 1980;
cout << "Year: " << year << endl;
```

Output

Code

Output

```
cout << "Hello!" << endl;
```

```
Hello!
```

```
cout << "Year: " << year << endl;
```

```
Year: 1980
```

```
cout << year << "-"
     << month << "-"
     << day << endl;
```

```
1980-3-1
```

Input

To get input from the user, we can use **`cin`** (console-in)

The **>>** signs after **`cin`** will point **towards** the variable, and away from the **`cin`**.

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int age;
7      cout << "What is your age? ";
8      cin >> age;
9
10     return 0;
11 }
```

Input

Code

```
int age;
cout << "What is your age? ";
cin >> age;

float money;
cout << "How much money do you have? $";
cin >> money;

char letter;
cout << "Favorite letter? ";
cin >> letter;
```

We can use **`cin`** with integers, floats, chars, strings, and other data types.

Output

```
What is your age? 265
How much money do you have? $9.53
Favorite letter? R
```

Input

The text is flowing ***towards*** the console-out

```
cout << "What is your age? ";
```

The input is flowing ***from*** the console-input,
towards the variable.

```
cin >> age;
```

The String Data Type

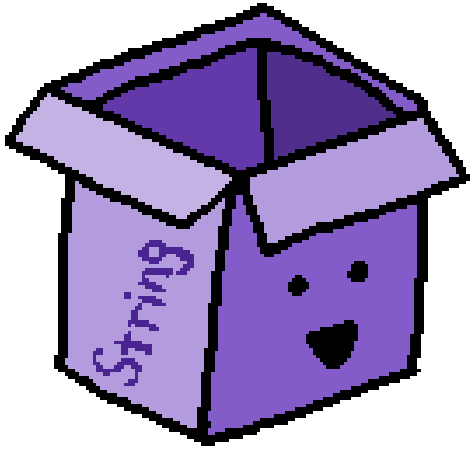
Written by Rachel J. Morris, last updated 2015-01-14

Data Types

The **`string`** data type requires that we include another library from the Standard Library.

```
#include <string>
```

A **`string`** is a ***string of characters*** – text, letters, numbers, anything.



Data Types

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

int main()
{
    string name;
    cout << "What is your name? ";
    cin >> name;
    cout << "Name: " << name << endl;

    return 0;
}
```

We can also use **`cin`** to get a string from the user.
However...

Data Types

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

int main()
{
    string name;
    cout << "What is your name? ";
    cin >> name;
    cout << "Name: " << name << endl;

    return 0;
}
```

```
What is your name? Fordon Greeman
Name: Fordon
```

The program only stores the first word here

Using **`cin`** this way only allows us to get any text, up to a blank space. Anything after the space may be passed to the next **`cin`** statement.

Types of Errors

Written by Rachel J. Morris, last updated 2015-01-14

Types of Errors

There are various types of errors that you can experience when programming.

Let's go over these, so that you can learn to look out for these problems.

Types of Errors

```
1  int main()
2  {
3      int a = 10;
4      intg b = 20;
5
6      return 0;
7  }
8
9
10
11
```

Logs & others

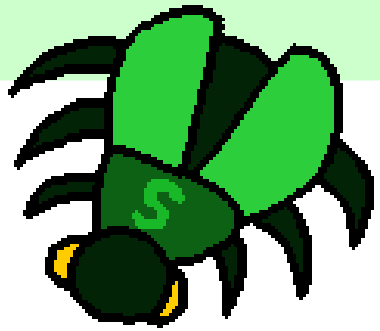
Build log | Build messages | Debugger

File	Line	Message
		=== Build: Debug in Sample Project (compiler: GNU GCC Comp
/home/rejcx/temp/...		In function 'int main()':
/home/rejcx/temp/...	4	error: 'intg' was not declared in this scope
/home/rejcx/temp/...	4	error: expected ';' before 'b'
/home/rejcx/temp/...	3	warning: unused variable 'a' [-Wunused-variable]
		=== Build failed: 2 error(s), 1 warning(s) (0 minute(s), 0

The first error is the real error – **intg** isn't a data type, it is a typo of **int**.

A **Syntax Error** is when something in the language has been incorrectly typed.

The compiler can easily detect these errors, and you will receive a **compile-time error**.

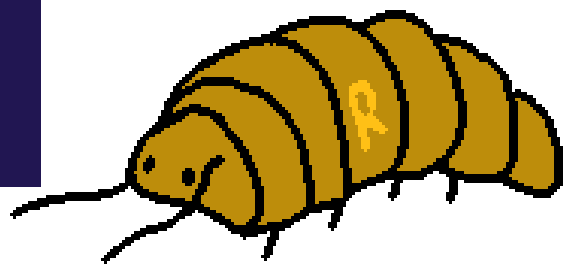


Types of Errors

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

int main()
{
    float prices[3] = { 9.99, 1.00, 2.50 };
    cout << "Price: $" << prices[5];

    return 0;
}
```



```
Price: $4.59163e-41
```

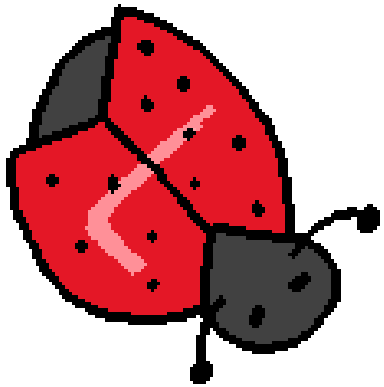
A **run-time** error is something that is not “grammatically incorrect” in your program, but something breaks (or doesn't work as intended) while the program is running.

Types of Errors

A **logic error** is an error that doesn't break the program, but doesn't work as intended – it is an error in logic. It could simply be getting a formula wrong.

```
int area = width + height;
```

Area is actually width x height



```
Dimensions: 20x30
Area: 50
```

When you run the program, it is clear that something is wrong, but you wouldn't know why without combing through the code.

Types of Errors

Syntax Error

The **Syntax Error** is easiest to find, since your program won't build unless your syntax is correct!

Run-Time Error

It might not be too difficult to spot **Run-Time Errors**, since your program may crash when you encounter one. Then you have to track down where it crashed...

Logic Error

Logic Errors are the hardest to diagnose, since the program will operate as written. The computer won't fact-check your formulas and algorithms for you. :)

So can we put it all together now?