C to C++ Transition

CSC 100 Intro to Programming in C++ (Spring 2025)

Marcus Birkenkrahe (pledged)

May 2, 2025

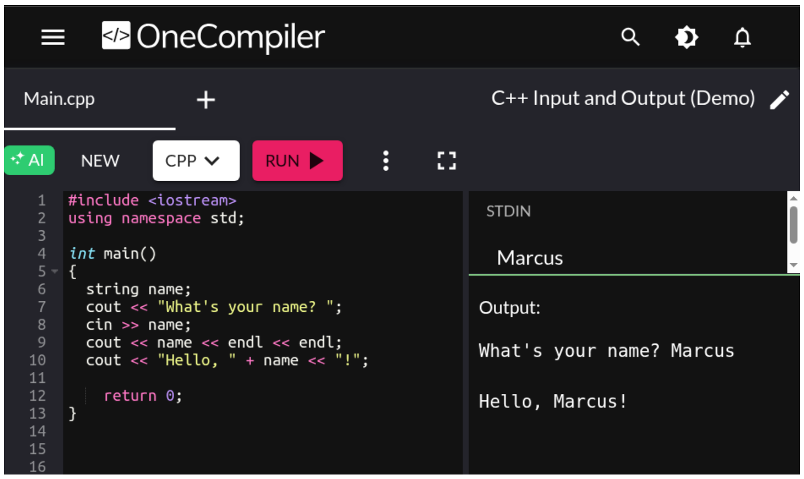
Codealong with C++ in OneCompiler

- OneCompiler is an IDE for multiple languages, and C/C++ are distinguished.
- Open `onecompiler.com/cpp` - this will bring up the template program for C++ rather than C - let's go through it line by line:

```
#include <iostream> // Input/output control
using namespace std; // Using stuff from the C++ standard library

int main(void) // the usual main program
{
  cout << "Hello, world!"; // direct the output to stdout
  return 0; // the usual END of main - 0 if successful
}
```

- It's just as (deceptively) simple to enter user input in C++. Replace the `cout` line with this and enter some input in the `STDIN` field.



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

int main()
{
  string name;
  cout << "What's your name? ";
  cin >> name;
  cout << name << endl << endl;
  cout << "Hello, " + name << "!";

    return 0;
}
```

STDIN

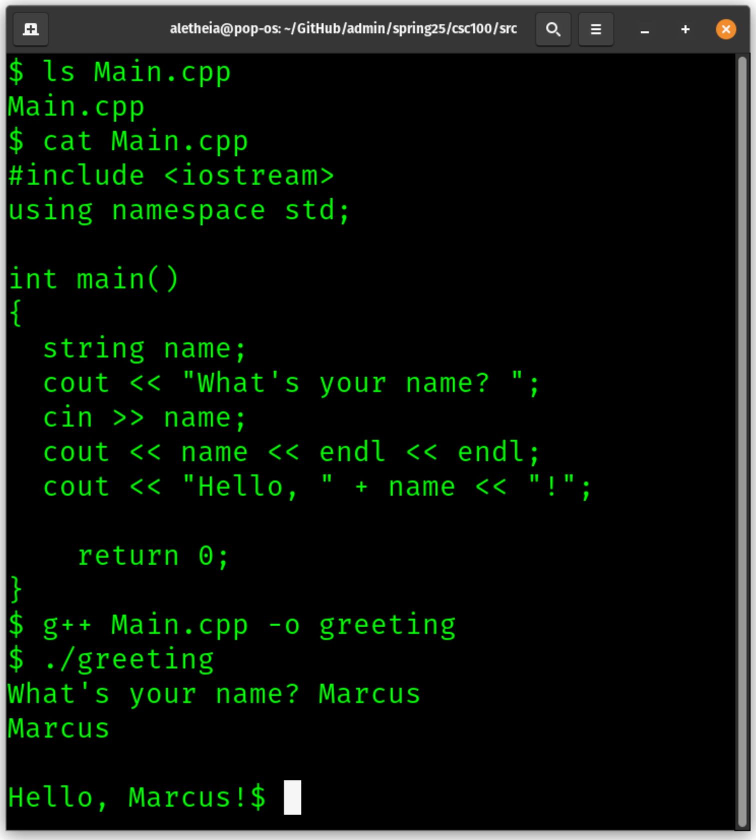
```
Marcus
```

Output:

```
What's your name? Marcus

Hello, Marcus!
```

- Notice that the program is called `Main.cpp` - .cpp is the default ending for C++ programs (compared to .c for C programs):
- You could download `Main.cpp`, compile and run it on the command-line (e.g. on the Google Cloud shell, `ide.cloud.google.com`):



```
$ ls Main.cpp
Main.cpp
$ cat Main.cpp
#include <iostream>
using namespace std;

int main()
{
  string name;
  cout << "What's your name? ";
  cin >> name;
  cout << name << endl << endl;
  cout << "Hello, " + name << "!";

    return 0;
}
$ g++ Main.cpp -o greeting
$ ./greeting
What's your name? Marcus
Marcus

Hello, Marcus!$
```

- The Code:

  ```
  string name; // declare string variable
  cout << "What's your name? "; // ask for user input
  cin >> name; // get user input from stdin
  cout << name << endl << endl; // print user input
  cout << "Hello, " + name << "!"; // print greeting
  ```

- **Explanations** (for lonely winter evenings):

  - `iostream` is the C++ header file for input and output
  - `std` is a `namespace`, a protected area that contains `cout` for output, `cin` for input, `string` and `vector` for string and vector identifiers, `endl` for new line, etc.

  - This is because there is a limited number of words and terms available, and different softwares can now use the same name but with a different `namespace` prefix.
  - The full version of `cout` is therefore `std::cout` etc.

- Input:

  ```
  echo "Marcus" > input
  cat input
  ```

  ```
  Marcus
  ```

From `struct` in C to `class` in C++ - `Player`

- Our goal is to understand how to translate a C `struct` into a C++ `class` and why you'd want to do that.
- Here is a typical C-style `struct`:

  ```
  struct Player {
    int x; // player's x position
    int y; // player's y position
    int health; // player's health points (> 0)
  };
  ```

- There's a problem here: In C, all members of `Player` are `public` by default: There's no way to restrict access if someone wants to mess with your `Player`.
- Example: Negative `Player.health` points are meaningless but the compiler allows it - and the position `Player.x` can also be corrupted:

  ```
  struct Player {
    int x; // player's x position
    int y; // player's y position
    int health; // player's health points (> 0)
  };

  struct Player John; // John is a Player now
  ```

```
John.health = -1000; // Invalid health!
John.x = 9999; // Out in the cold!

printf("Player's health (%d) and position (%d) are worrying.\n",
       John.health, John.x);
```

```
Player's health (-1000) and position (9999) are worrying.
```

Practice: From `struct` in C to `class` in C++ - `Date`

1. Turn the following C `struct Date` into a C++ `class`. Do this by changing the keyword `struct` to `class`, and putting `public:` in the first line of the `class`.

   ```
   struct Date {
     int day;
     int month;
     int year;
   };
   ```

2. In the `main` program, create a `Date` called `today` and assign it today's date as `{[day],[month],[year]}`, then print `today`.

3. Sample output:

   ```
   Today's date: 2/5/2025
   ```

4. Starter code: onecompiler.com/cpp/43gma9be6

   ```
   // include input/output stuff

   // create Date class with day, month, year (int)
   // BEGIN CLASS
     public: // public data
       // day
       // month
       // year
   // END CLASS
   ```

```
// main program
int main(void)
{
  // declare and initialize today as a Date

  // PRINT today's date

  return 0;
}
```

Solution:

- Code:

```
// include input/output stuff
#include <iostream>
// create Date class with day, month, year (int)
class Date {
public:
  int day;
  int month;
  int year;
};
// main program
int main(void)
{
  // declare and initialize today as a Date
  Date today = {2,5,2025};
  // PRINT today's date
  printf("Today's date: %d/%d/%d\n",
     today.day, today.month, today.year);
  return 0;
}
```

- Explanation for `public`:

  > Members of a C++ `class` are automatically `private` and cannot be accessed from outside the class unless they are made `public`.

Data hiding (aka encapsulation) in C++ - `Player.health`

- In C++, `Player` data can be hidden and controlled:

```
class Player {

private:      // private data
  int x, y, health;
};
```

- Let's try to mess with a player now:

```
class Player {

private:      // private data
  int x, y, health;
};

class Player Jane;
Jane.health = -1000;
Jane.x = 9999;
```

- In C++, if you don't specify data as `public`, they're `private`.

Practice: `Date` class

- Use the `Date class` code defined and used earlier.
- In the `class` declaration, make the data `private`.
- In the `main` program, only create a `Date` for `today` and initialize it with today's date.
- What's the output?

  > Permission to initialize is denied, because the data are `private` to `today`. The error message is somewhat cryptic. `Date today;` works.

- Solution:

```
// include input/output stuff
#include <iostream>
// create Date class with day, month, year (int)
class Date {
  int day;
  int month;
  int year;
};
// main program
int main(void)
{
  // declare and initialize today as a Date
  Date today = {2,5,2025};
  return 0;
}
```

How to access `private` data - `getHealth()`

- Data that are `private` are accessed only indirectly through **methods**.
- You've already met one one those methods: `move_point` for the `Point` structure:

  ```
  struct Point p;  // create a Point p
  move_point(&p,dx,dy); // move p by dx in x-, and by dy in y-direction
  ```

- Methods are functions that belong to classes and act on their data. In C++, a method (or member function)
  1. is declared inside a `class`
  2. can access the class's `private` data
  3. is called using an object of the `class`
- Here's `Player` again but with a method that allows us to check the `Player`'s `health`:

  ```
  class Player {
  private:
    int health = 100;  // Player's private health
  public:
  ```

```
  int getHealth() {
    return health;  // make Player's health public
  }
};
```

- Let's test it:

```
class Player {
private:
  int health = 100;  // Player's private health
public:
  int getHealth() {
    return health;  // make Player's health public
  }
};
 // Create a Player named Jane
class Player Jane;
// Get Jane's [private] health data
cout << "Player health = " << Jane.getHealth() << endl;
```

- C++ enforces data type and access control much more strongly than C.

Practice: Get the `Date` for `today` with `getDate()`

- Add a method `printDate` to the `Date class` with so that you can print today's date.

- Solution:

```
// include input/output stuff
#include <iostream>
// create Date class with day, month, year (int)
class Date {
public:
  int day;
  int month;
  int year;
  void printDate() {
    printf("Today's date: %d/%d/%d\n",day, month, year);
  }
};
// main program
int main(void)
{
  // declare and initialize today as a Date
  Date today = {2,5,2025};
  today.printDate();
  return 0;
}
```

How to alter `private` data - `takeDamage`

- Now we know how to get to the `private` data - to alter them, we need a new method. In the example, we're adding the `takeDamage` method, and we're retaining the `getHealth` method (we need it to check).

- Example: Create a `Player` that can take damage

```
class Player { // a Player class

private: // private data
  int health = 100;  // Player's health is hidden

public:  // public member function

  int getHealth() {
    return health;  // make Player's health public
  }

  void takeDamage(int amount) {
    health -= amount; // reduce Player's health by amount
  }
};
```

- In the `main` program, we're adding a `Player` who can take damage:

```
class Player { // a Player class

private: // private data
```

```
  int health = 100;  // Player's health is hidden

public:  // public member function

  int getHealth() {
    return health;  // make Player's health public
  }

  void takeDamage(int amount) {
    health -= amount; // reduce Player's health by amount
  }
};

class Player John; // John's a Player

// What's his health like?
cout << "Before the fight: Player's health = " << John.getHealth() << endl;

// In a fight, John takes damage
John.takeDamage(50);

// What's his health like?
cout << "After the fight: Player's health = " << John.getHealth() << endl;
```

Challenge: Heal the `Player` with `heal`

- Use the code developed so far, and add a `heal` method that increases a `Player`'s `health`:
  1. Create `Player` class with `private` member `health`, and `public` methods `getHealth`, `takeDamage`, and `heal`.
  2. Create `main` program, create a `Player`, print his `health`, let him `takeDamage` (50), print his `health`, `heal` him (80), print `health`.
- Sample output:

```
Player's health = 100
Player's health after battle = 50
Player's health after healing = 130
```

- Here is the starter code:

```
// include input / output
// use standard names

/* class definition */
// Create a Player class


// private data

// Player's health (initially 100)

// public data

// Return Player's health
// int getHealth(void)

// Reduce Player's health by amount
// void takeDamage(int)

// Heal Player by amount
// void heal(int)

// END CLASS

/* main program */

// BEGIN MAIN
// Create a Player [name]

// PRINT Player's health + new line

// Player takes damage (50)

// PRINT Player's health after battle + new line

// Player heals (80)

// PRINT Player's health after healing + new line
// END MAIN
```

- Solution:

```
#include <iostream> // include input / output
using namespace std; // use standard names

// Create a Player class
class Player {

private: // private data
  // Player's health (initially 100)
  int health = 100;

public: // public data

  // Return Player's health (int)
  int getHealth() {
    return health;
  }

  // Reduce Player's health by amount (int)
  void takeDamage(int amount) {
    health -= amount;
  }
  // Heal Player by amount (int)
  void heal(int amount) {
    health += amount;
  }
};

/* main program */
int main(void)
{
  // Create a Player
  Player John;
  // PRINT Player's health
  cout << "Player's health = " << John.getHealth() << endl;
  // Player takes damage (50)
  John.takeDamage(50);
  // PRINT Player's health after battle
  cout << "Player's health after battle = " << John.getHealth() << endl;
  // Player heals (80)
  John.heal(80);
  // PRINT Player's health after healing
  cout << "Player's health after healing = " << John.getHealth();

  return 0;
}
```

Bonus challenge

- Modify the previous program to cap the `health` at 100. That is, `if` `health` is above 100, reset it to 100.
- Using the same values as before (take 50 damage, heal 80), the sample output is now:

  ```
  Player's health = 100
  Player's health after battle = 50
  Player's health after healing = 100
  ```

- Solution:

  ```
  #include <iostream> // include input / output
  using namespace std; // use standard names

  // Create a Player class
  class Player {

  private: // private data
    // Player's health (initially 100)
    int health = 100;

  public: // public data

    // Return Player's health (int)
    int getHealth() {
      return health;
  ```

```
  }

  // Reduce Player's health by amount (int)
  void takeDamage(int amount) {
    health -= amount;
  }
  // Heal Player by amount (int)
  void heal(int amount) {
    health += amount;
    if (health > 100) health = 100;
  }
};

/* main program */
int main(void)
{
  // Create a Player
  Player John;
  // PRINT Player's health
  cout << "Player's health = " << John.getHealth() << endl;
  // Player takes damage (50)
  John.takeDamage(50);
  // PRINT Player's health after battle
  cout << "Player's health after battle = " << John.getHealth() << endl;
  // Player heals (80)
  John.heal(80);
  // PRINT Player's health after healing
  cout << "Player's health after healing = " << John.getHealth();

  return 0;
}
```