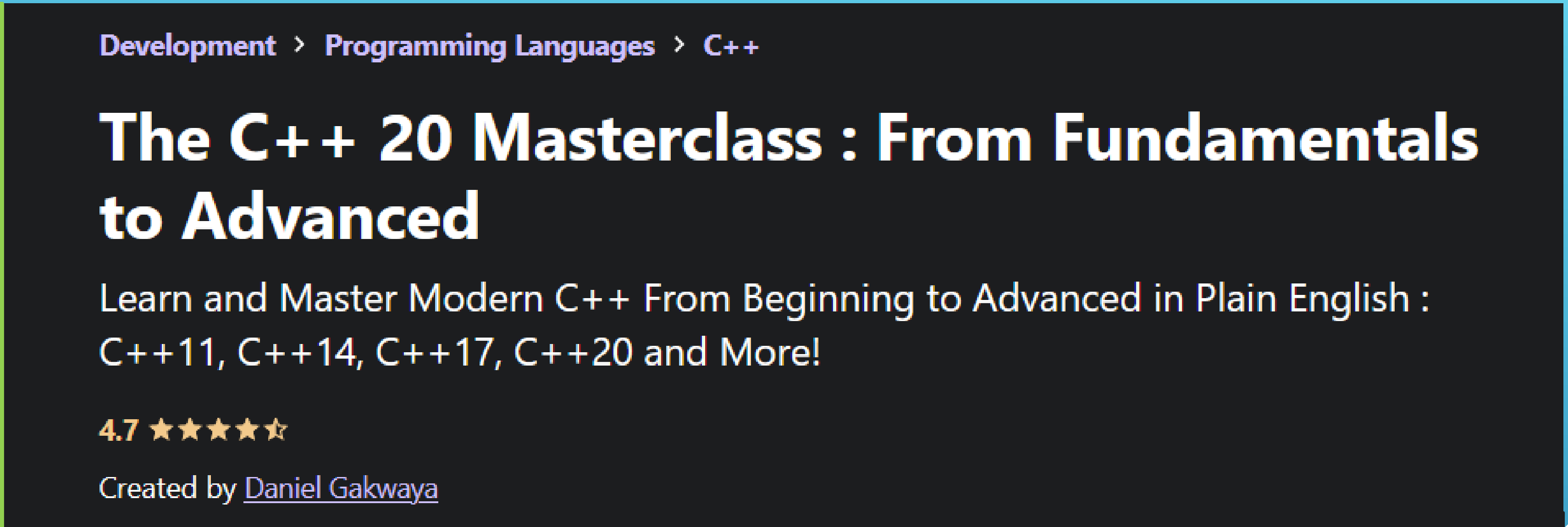
Slides


Development > Programming Languages > C++

The C++ 20 Masterclass : From Fundamentals to Advanced

Learn and Master Modern C++ From Beginning to Advanced in Plain English : C++11, C++14, C++17, C++20 and More!

4.7 ★★★★☆

Created by Daniel Gakwaya

Section : Functions – The misfits

Slide intentionally left empty

Functions : The misfits

Static variables

Inline functions

Recursive functions

Slide intentionally left empty

Static variables

Static variables

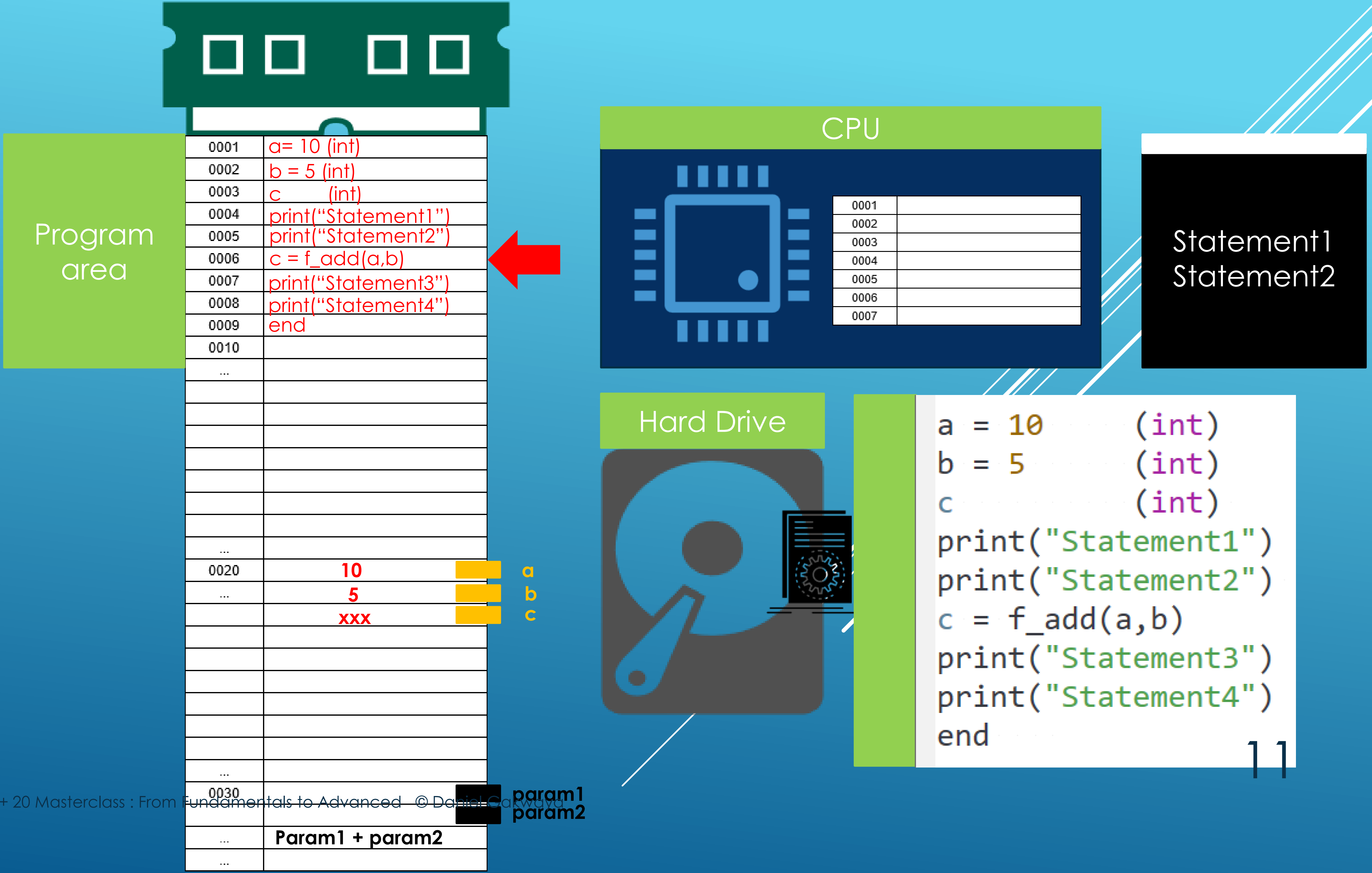
```cpp
int student_count_global{0};

int add_student(){
    static int student_count{0}; // The scope of this var is in add_student,
                                // but it's life time goes beyond the function execution.
    student_count++;
    return student_count;
}

int add_student_using_global_var(){
    student_count_global++;
    //student_count++;  // Compiler error, this student_count can only
                        // be used in the add_student function.
                        // It has function local block scope
    return student_count_global;
}

void some_other_function(){
    student_count_global--;
}

int main(int argc, char **argv)
{
    return 0;
}
```

Global variable vs Static variable

- Both global and static variables have static storage duration. They live throughout the entire lifetime of the program
- Static variables are scoped to the function in which they are declared and used. If you try to access them outside that function, you'll get a compiler error
- Global variables are scoped to the global scope of the file where they are declared. They are accessible and usable through out the entire file.

Slide intentionally left empty

Inline functions



```
a = 10      (int)
b = 5       (int)
c           (int)
print("Statement1")
print("Statement2")
c = f_add(a,b)
print("Statement3")
print("Statement4")
end
```

Regular non inline

```
#include <iostream>

int max(int a, int b){
    if(a> b){
        return a;
    }else{
        return b;
    }
}

int main(int argc, char **argv)
{
    int value1{34};
    int value2{60};

    //Without inline,
    std::cout << "max : " <<  max(value1,value2) << std::endl;

    /* ...

    return 0;
}
```

Inline

```
inline int max(int a, int b){
    if(a> b){
        return a;
    }else{
        return b;
    }
}

int main(int argc, char **argv)
{
    int value1{34};
    int value2{60};

    std::cout << "max : " <<  max(value1,value2) << std::endl;






    return 0;
}
```

Inline

```
inline int max(int a, int b){
    if(a> b){
        return a;
    }else{
        return b;
    }
}

int main(int argc, char **argv)
{
    int value1{34};
    int value2{60};

    std::cout << "max : " <<  max(value1,value2) << std::endl;




    return 0;
}
```

Inline

```
inline int max(int a, int b){
    if(a> b){
        return a;
    }else{
        return b;
    }
}

int main(int argc, char **argv)
{
    int value1{34};
    int value2{60};

    ~~std::cout << "max : " << max(value1,value2) << std::endl;~~




    return 0;
}
```

Inline

```
inline int max(int a, int b){
    if(a> b){
        return a;
    }else{
        return b;
    }
}

int main(int argc, char **argv)
{
    int value1{34};
    int value2{60};

    ~~std::cout << "max : " << max(value1,value2) << std::endl;~~

    std::cout << "max (" << value1 << "," << value2 << ") : ";
    if(value1 > value2){
        std::cout << value1 ;
    }else{
        std::cout << value2;
    }
    std::cout << std::endl;
    return 0;
}
```

- Inline functions can increase the size of your application binary
- It is recommended to use them for short, frequently used functions
- The programmer (You), should weigh in the benefits against the downsides of inlining your functions
- Usually only functions of a few lines of code and simple logic, like our max function should be inlined
- Marking your function as inline is just a suggestion to the compiler. The compiler might agree and inline your function or just ignore you

Slide intentionally left empty

Recursive Functions

Recursion

A mechanism under which a function repeatedly calls itself to achieve some goal.
A function that does recursion is called a recursive function.

Recursion in action

```
#include <iostream>

int sum_up_to_zero(int value){
    if(value!=0)
        return value + sum_up_to_zero(value-1);
    return 0;
}

int main(int argc, char **argv)
{
    std::cout << "sum : " << sum_up_to_zero(3) << std::endl;
    return 0;
}
```

Slide intentionally left empty

Functions - The misfits : Summary

Static variables

Inline functions

Recursive functions