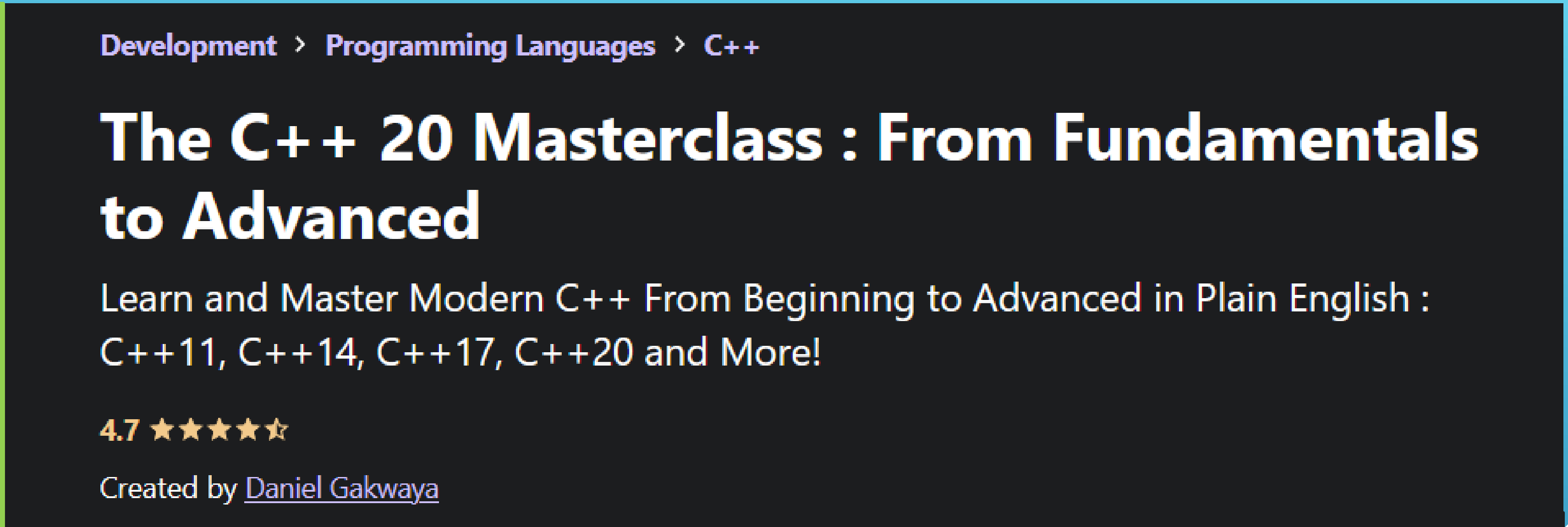
Slides



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!

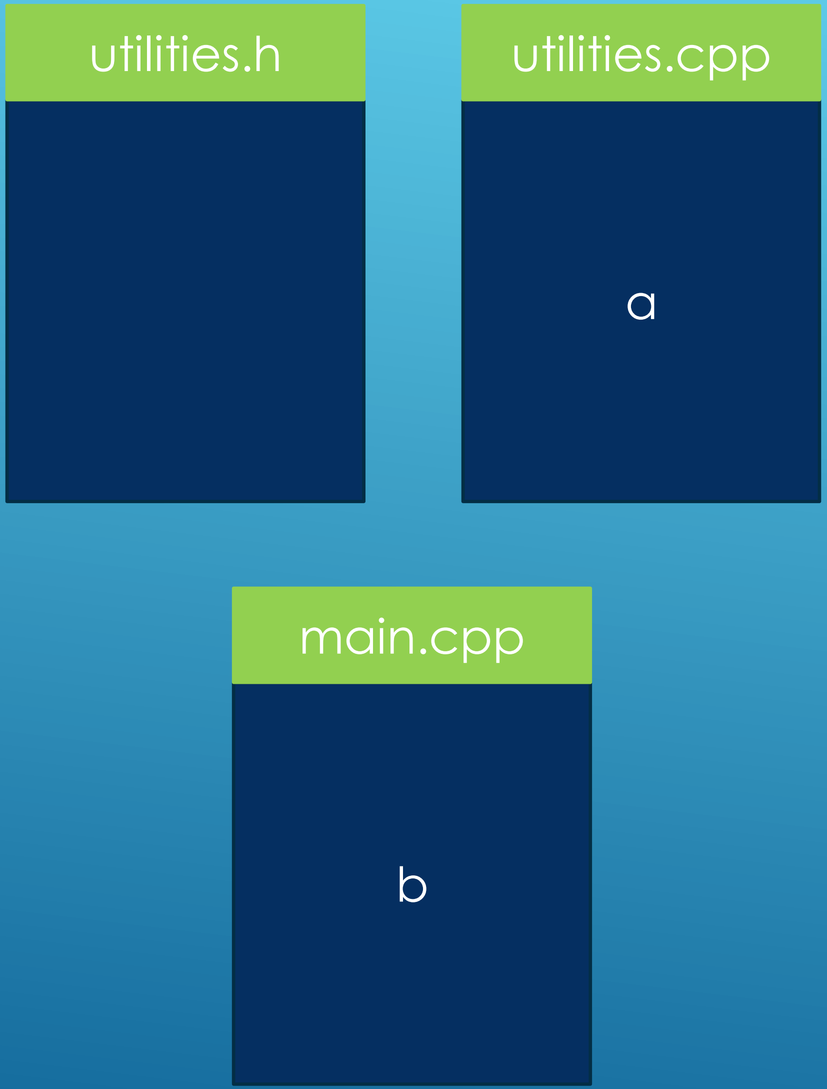
4.7 ★★★★★

Created by Daniel Gakwaya

Section : Programs with multiple files – A closer look

Slide intentionally left empty

Programs with Multiple Files : A closer look

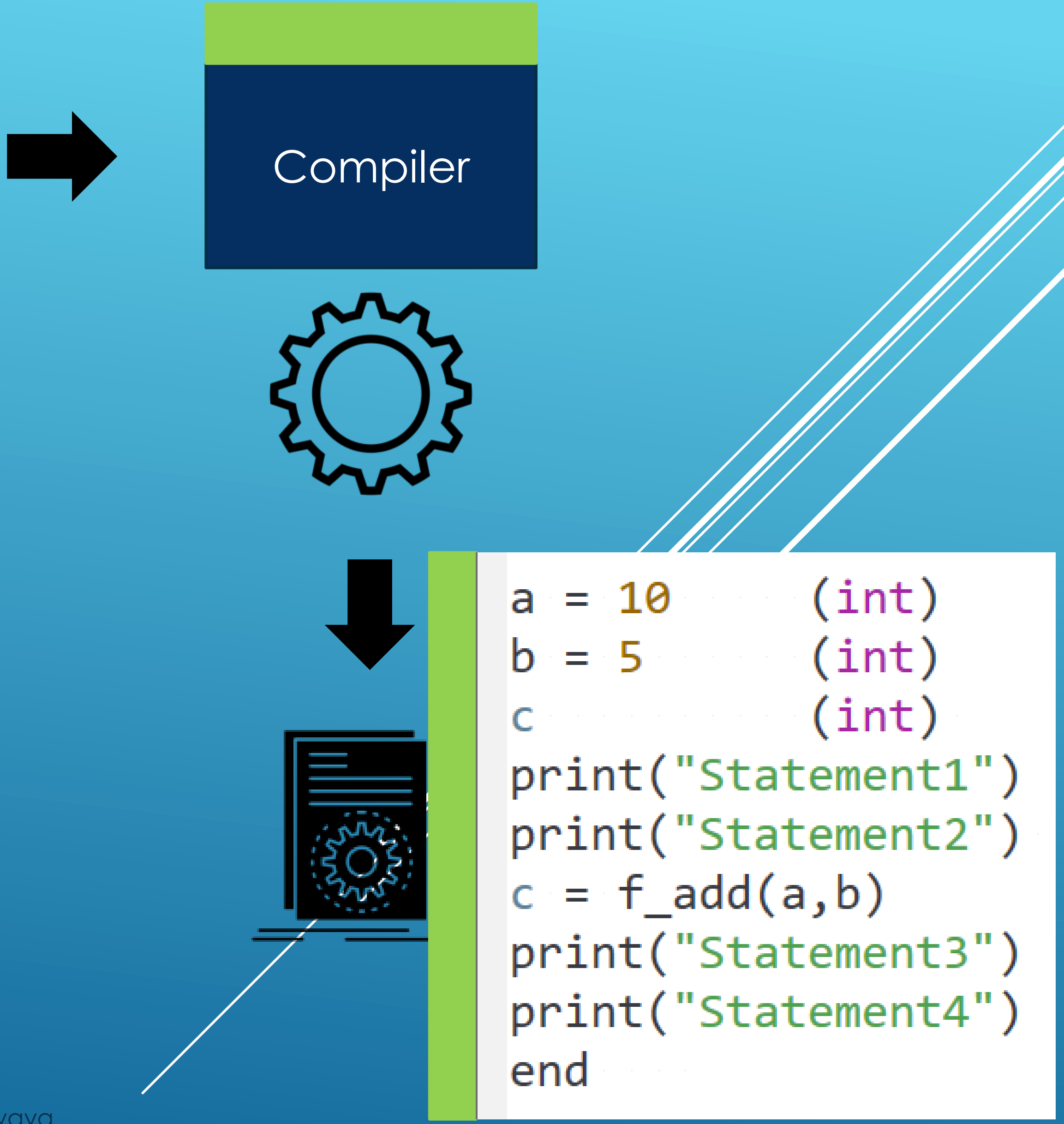

utilities.h
utilities.cpp
a
main.cpp
b

Slide intentionally left empty

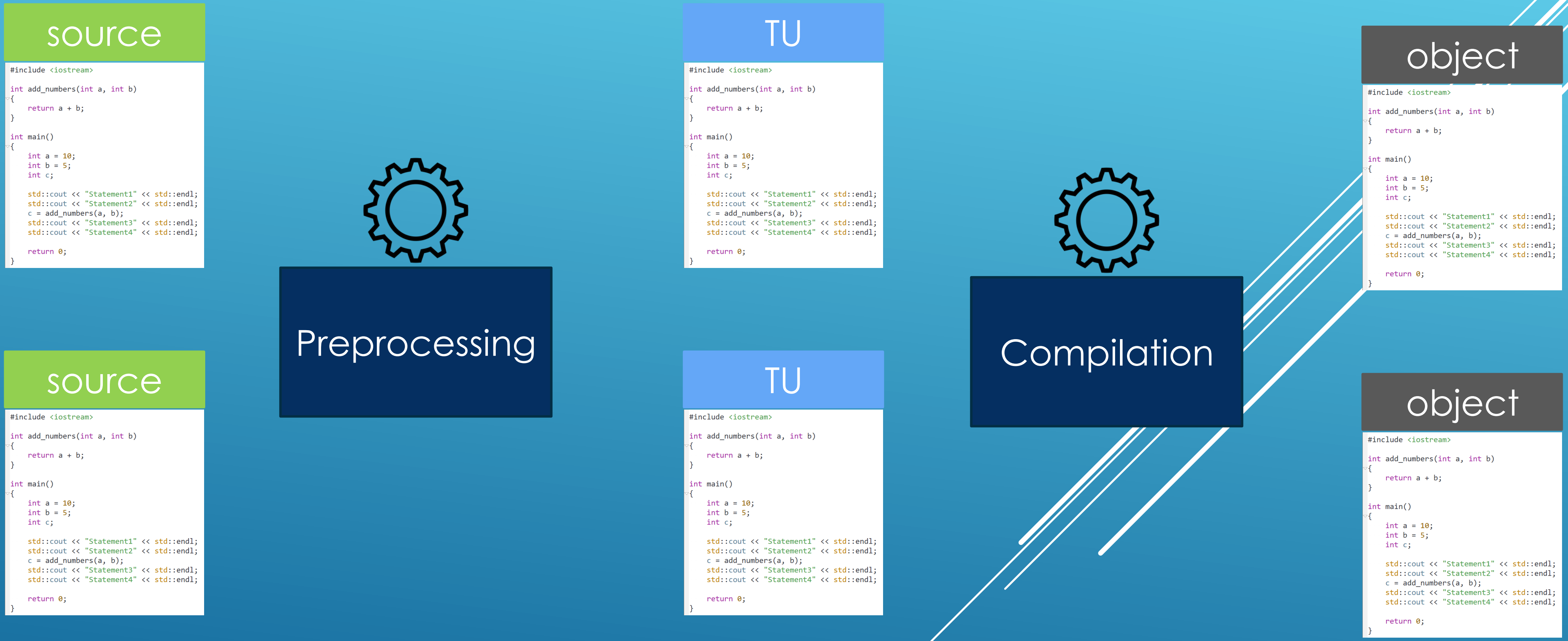
C++ Compilation model

One file program

```cpp
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```



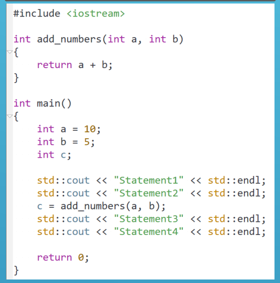
```
a = 10        (int)
b = 5         (int)
c             (int)
print("Statement1")
print("Statement2")
c = f_add(a,b)
print("Statement3")
print("Statement4")
end
```



```
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

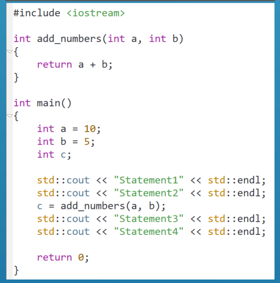
```
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

object



```
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

object



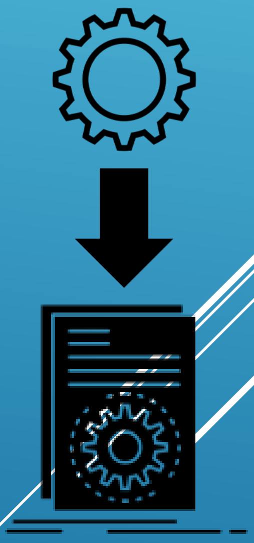
```
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

Linking

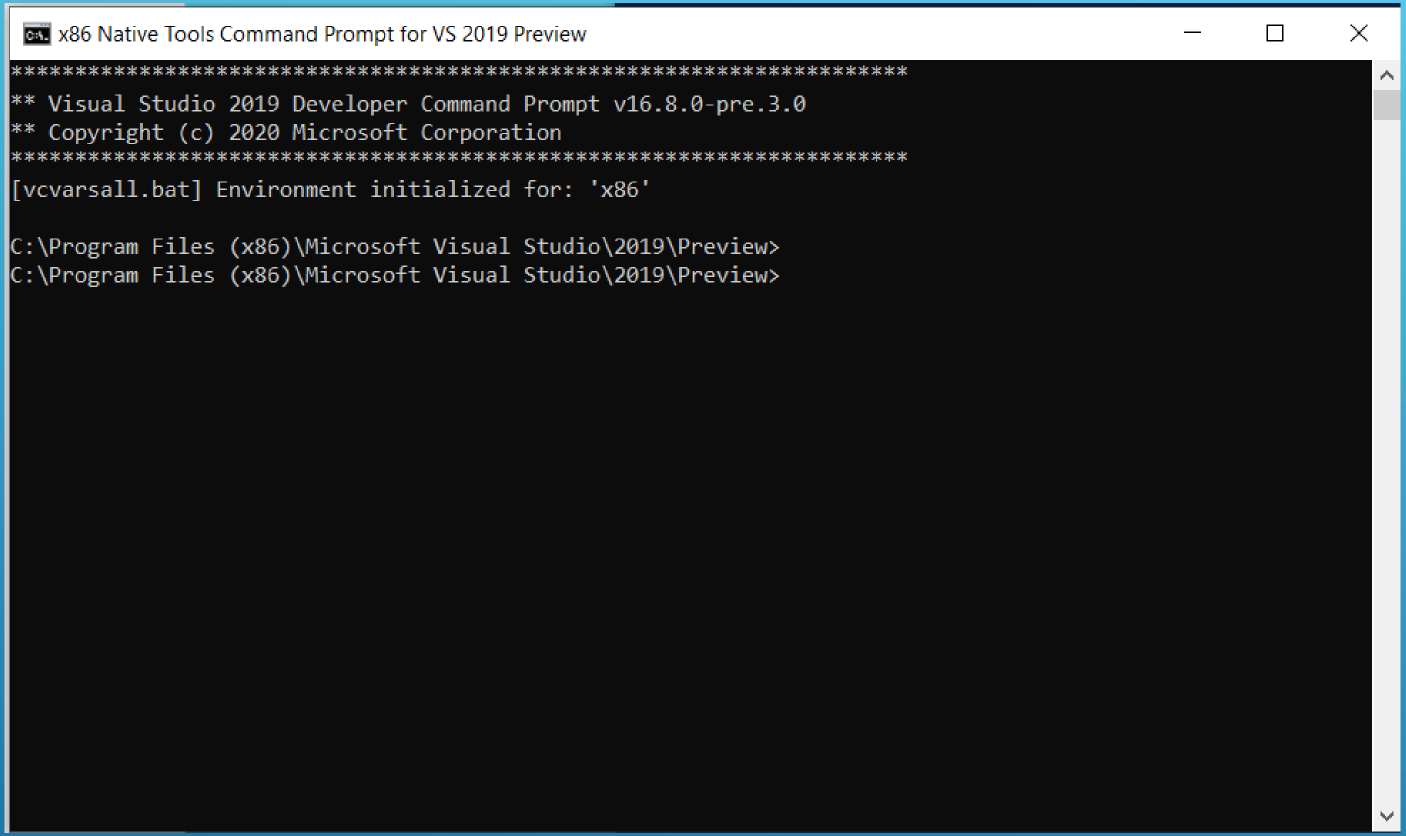
One file program

```cpp
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

- Preprocessing
- Compilation
- Linking



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



```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.8.0-pre.3.0
** Copyright (c) 2020 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86'

C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview>
C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview>
```

IDEs actually call compilers behind the scenes for us to compile our code and generate an binary we can run. In this lecture, we'll see how we can bypass the IDE and do all this ourselves.

GCC will be our compiler in the course but the concepts apply to any C++ compiler out there : MSVC, Clang,…

utilities.h
utilities.cpp
main.cpp

Compile and link in one go!

```
g++ -o rooster.exe main.cpp utilities.cpp
```

Compile only : generate object files

```
g++ -c main.cpp utilities.cpp
```

Link up object files

```
g++ -o rooster.exe main.o utilities.o
```

Compiler path should be in system environment variables

Slide intentionally left empty

Declarations and definitions

Declaration introduces the name in a file, a definition says what that name is or what it does.

```
//Function declaration
void some_function();

int main(int argc, char **argv)
{
    some_function();

    /* ...
}

//Function definition
void some_function(){

}
```

A few facts about declarations & definitions

- If a name is never used (function called, or variable read from/written to) in main, it's definition won't be needed : code will compile just fine.
- If you compile without a declaration, and the name is used, you get a compiler error : unkown name
- If you compile with declaration without definition and the name is not used,code compiles
- If you compile with declaration, without definition and the name is used, you get a linker error.

- Variables
- Functions
- Classes

Variables

```
//Declaration doubles as definition
double weight{};
```

Functions

```cpp
//Function declaration
void some_function();
```

Functions

```
//Function definition
void some_function(){

}
```

Classes

```
//Doubles as declaration and definition.
// Commonly refered to as definition
struct Point{
    double x;
    double y;
};
```

Classes

```cpp
class Person{
public :
    Person(const std::string& names_param, int age_param);
private :
    std::string full_name;
    int age;
};
```

Slide intentionally left empty

One Definition Rule

Definitions can’t show up more than once in entire program, or Translation units.

One definition rule : context

- Free standing variables
- Functions
- Classes
- Class member functions
- Class static member variables

```cpp
class Person{
public :
    Person(const std::string& names_param, int age_param);

    void print_info()const{
        std::cout << "name : " << full_name << " , age : " << age << std::endl;
    }
private :
    std::string full_name;
    int age;
};
```

Slide intentionally left empty

Linkage

A property associated with a name, that controls how wide or narrow is the visibility of the name across translation units

No linkage

```
void some_function(){
    int age{6}; // Age has no linkage
    std::cout << "age : " << age << std::endl;
}
```

Internal linkage

```
const double distance{43.7};// distance has internal linkage
                             // only visible in this TU
                             // If you try to declare it in another TU, you
                            // will be creating a completely separate new
                            // definition
```

Extern linkage

```
int item_count{7}; // Extern linkage

void print_distance(); // Extern linkage
void print_item_count();
```

Linkage options

- No linkage
- Internal Linkage
- External Linkage
- Module Linkage(later)

Linkage defaults

- Function local variables have no linkage
- Const global variables have internal linkage
- Non const global variables have external linkage
- Functions have external linkage

Slide intentionally left empty

Global variables

main.cpp

```cpp
#include <iostream>

int age {12}; // Global variable with external linkage

const double distance {44.9};

void print_age(); // Declaration
void print_distance(); // Declaration


int main(int argc, char **argv)
{
    std::cout << "non const global variables (extern linkage) : " << std::endl;
    std::cout << "age(main) : " << age << std::endl;
    std::cout << "&age(main) : " << &age << std::endl;
    print_age();

    std::cout << std::endl;
    std::cout << "const global variables(internal linkage) : " << std::endl;
    std::cout << "distance(main) : " << distance << std::endl;
    std::cout << "&distance(main) : " << &distance << std::endl;
    print_distance();
    return 0;
}
```

some_other_file.cpp

```
#include <iostream>
extern int age; // Declaration

const double distance{};// This is not a declaration, it's a definition of a completely
                        // new and separate variable. Can take a look at the addresses
                        // to confirm this.
//Definition
void print_age(){
    std::cout << "age(main) : " << age << std::endl;
    std::cout << "&age(main) : " << &age << std::endl;
}


void print_distance(){
    std::cout << "distance(main) : " << distance << std::endl;
    std::cout << "&distance(main) : " << &distance << std::endl;
}
```

Slide intentionally left empty

Flipping Linkage

main.cpp

```cpp
#include <iostream>

extern const double distance {45.9}; // Declaration & definition

void print_distance();// Declaration
void some_function(); // Declaration

int main(int argc, char **argv)
{
    std::cout << "distance(main) : " << distance << std::endl;
    std::cout << "&distance(main) : " << &distance << std::endl;

    std::cout << std::endl;
    print_distance();

    std::cout << std::endl;
    some_function();
    return 0;
}
```

some_other_file.cpp

```cpp
#include <iostream>

extern const double distance; // Declaring const double variable, defined in
                              // some_other_file

void print_distance(){
    //++distance; // Error : can't modify a read only variable.
    std::cout << "distance(some_other_file) : " << distance << std::endl;
    std::cout << "&distance(some_other_file) : " << &distance << std::endl;
}

/*
static void some_function(){
    std::cout << "some_function called ..." << std::endl;
}
*/

namespace{
    void some_function(){
        std::cout << "some_function called ..." << std::endl;
    }
}
```

Slide intentionally left empty

Inline variables and functions

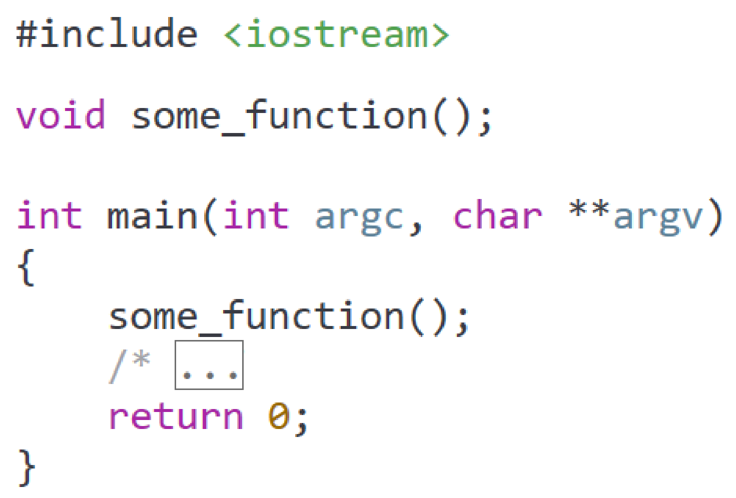
utility1.cpp

```
#include <iostream>

 int age{12};

 void some_function(){
     std::cout << "age : " << age << std::endl;
     std::cout << "&age : " << &age << std::endl;
}
```

utility2.cpp

```
#include <iostream>

 int age{12};

 void some_function(){
     std::cout << "age : " << age << std::endl;
     std::cout << "&age : " << &age << std::endl;
}
```

main.cpp



```cpp
#include <iostream>

void some_function();

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

utility1.cpp

```
#include <iostream>

inline int age{12};

inline void some_function(){
    std::cout << "age : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}
```

utility2.cpp

```
#include <iostream>

inline int age{12};

inline void some_function(){
    std::cout << "age : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}
```

utility1.cpp

```cpp
#include <iostream>

inline int age{12};

inline void some_function(){
    std::cout << "age : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}

void print_age_1(){
    some_function();
}
```

utility.h

```cpp
#include <iostream>

inline double threashold {11.1};


inline double add(double a, double b){
    if( (a > 11.1) && (b > 11.1)){
        return a + b;
    }else{
        return threashold;
    }
}
```

Slide intentionally left empty

Inline VS static (unnamed namespaces)

- Inline will optimize all the definitions for a name into one
- Static or unnamed namespaces won’t do such optimizations

utility1.cpp

```cpp
inline int age{12};

inline void some_age_function(){
    std::cout << "age  : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}

void print_age_1(){
    std::cout << "age(utility1) : " << std::endl;
    some_age_function();
}
/* ...
namespace{
    double distance {23.9};

    void some_distance_function(){
        std::cout << "distance : " << distance << std::endl;
        std::cout << "&distance : " << &distance << std::endl;
    }
}

void print_distance_1(){
    std::cout << "distance(utility1) : " << std::endl;
    some_distance_function();
}
```

utility2.cpp

```
inline int age{12};

inline void some_age_function(){
    std::cout << "age : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}

void print_age_2(){
    std::cout << "age(utility2) : " << std::endl;
    some_age_function();
}
/* ...
namespace{
    double distance {23.9};

    void some_distance_function(){
        std::cout << "distance : " << distance << std::endl;
        std::cout << "&distance : " << &distance << std::endl;
    }
}
void print_distance_2(){
    std::cout << "distance(utility2) : " << std::endl;
    some_distance_function();
}
```

main.cpp

```cpp
//Inline
void print_age_1();
void print_age_2();

//static (unamed namespaces)
void print_distance_1();
void print_distance_2();

int main(int argc, char **argv)
{
    std::cout << std::endl;
    std::cout << "inline : " << std::endl;
    print_age_1();
    std::cout << std::endl;
    print_age_2();

    std::cout << std::endl;
    std::cout << "static : " << std::endl;
    print_distance_1();
    std::cout << std::endl;
    print_distance_2();
    return 0;
}
```

Slide intentionally left empty

Forward Declarations

```cpp
class Dog
{
public:
    Dog() = default;
    Dog(const std::string& name);
    ~Dog();

    void print_info() const{
        std::cout << "Dog [ name : " << name << "]" << std::endl;
    }
private :
    std::string name{};
};
```

```
class Farm
{
public:
    Farm();
    ~Farm();

    void use_dog(const Dog& dog_param); // Doesn't require the definition
};
```

```cpp
#include "dog.h"

class Farm
{
public:
    Farm();
    ~Farm();

    void use_dog(const Dog& dog_param); // Doesn't require the definition
};
```

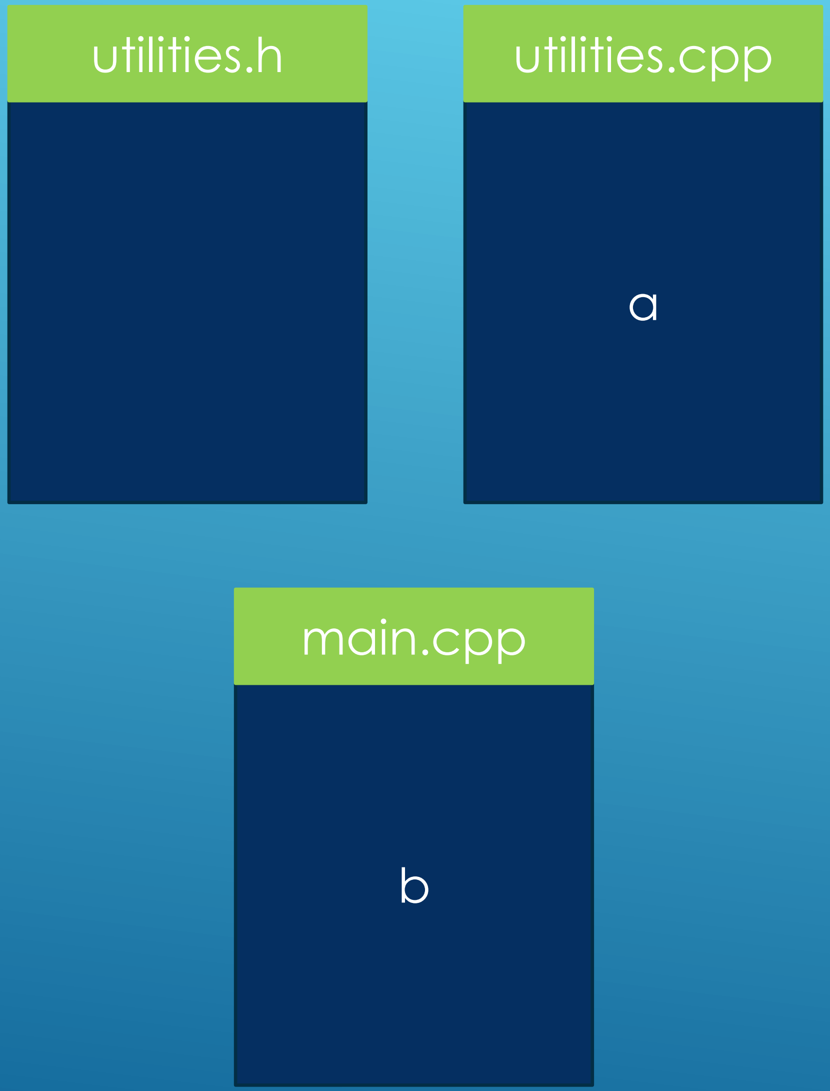
```cpp
class Dog; //Forward declaration

class Farm
{
public:
    Farm();
    ~Farm();

    void use_dog(const Dog& dog_param); // Doesn't require the definition
};
```

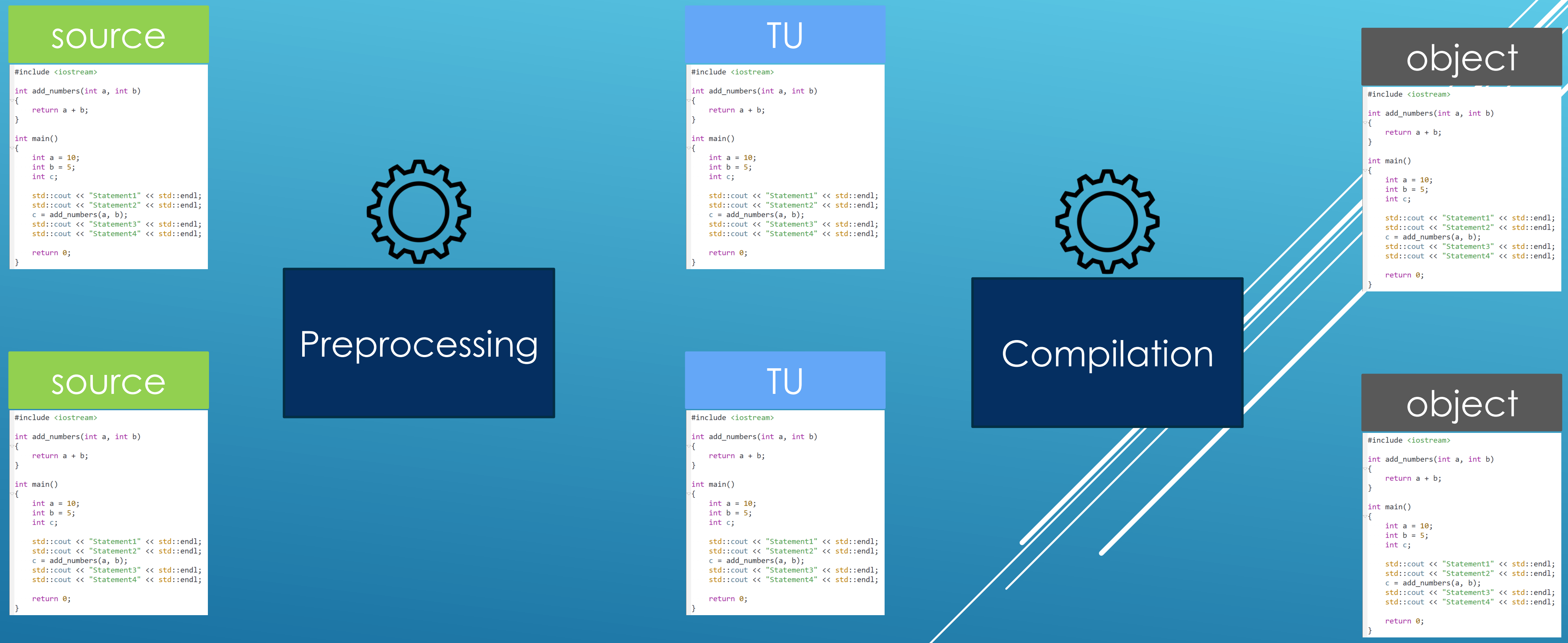
Sometimes real definitions are needed

```cpp
class Farm
{
public:
    Farm();
    ~Farm();

    void use_dog(const Dog& dog_param); // Doesn't require the definition

    //Definition of Dog needed : Forward declaration won't work
    void use_dog(const Dog& dog_param){
        dog_param.print_info();
    }

private :
    Dog guard; //Definition of Dog needed : Forward declaration won't work
};
```

Slide intentionally left empty

Programs with Multiple Files - A closer look :Summary

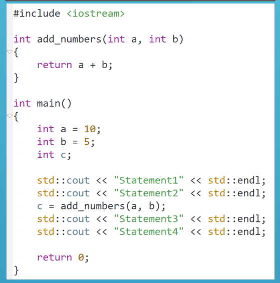

utilities.h
utilities.cpp
a
main.cpp
b



```cpp
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

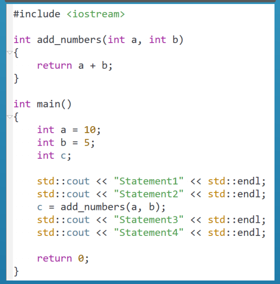
```cpp
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

object



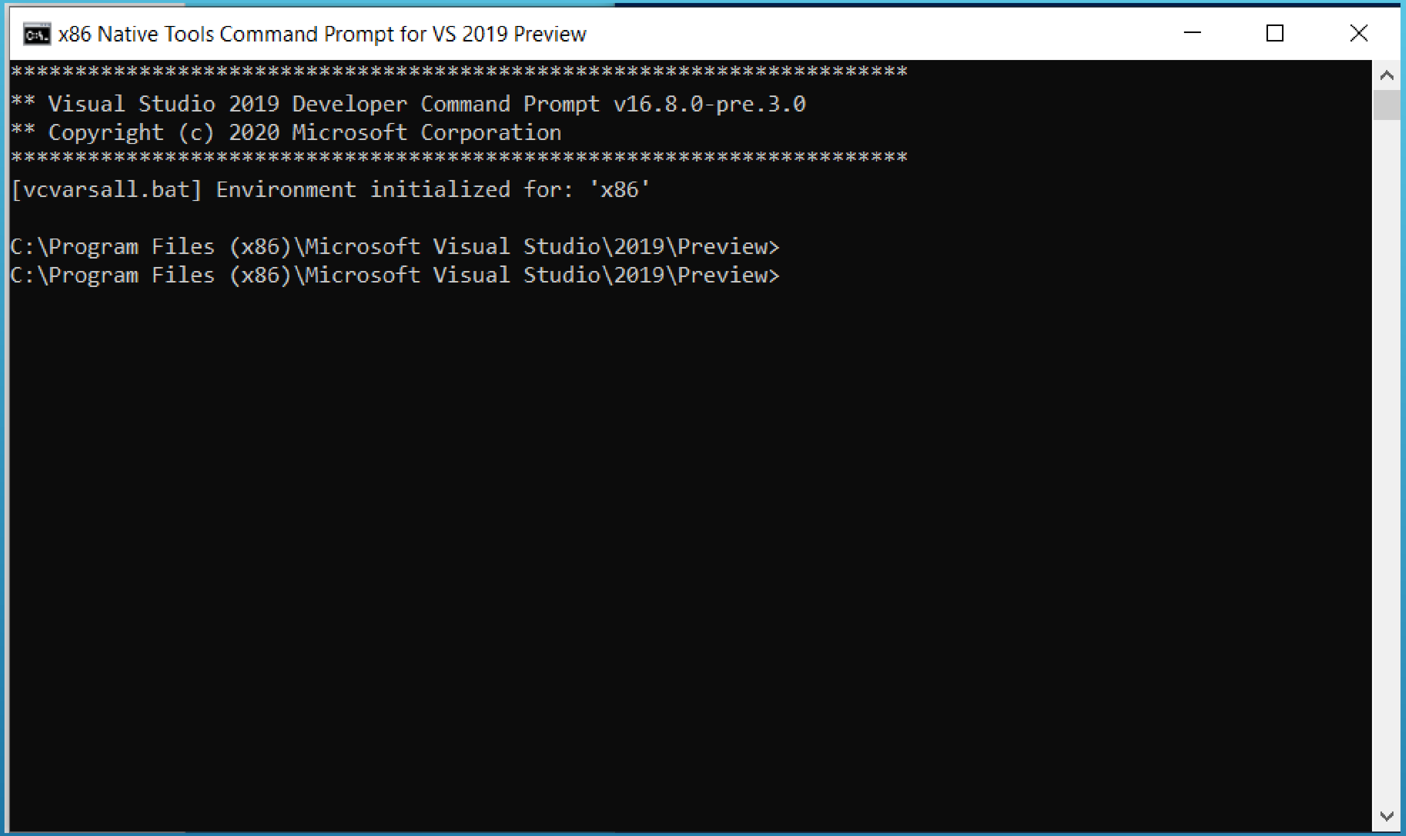
```cpp
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

object



```cpp
#include <iostream>

int add_numbers(int a, int b)
{
    return a + b;
}

int main()
{
    int a = 10;
    int b = 5;
    int c;

    std::cout << "Statement1" << std::endl;
    std::cout << "Statement2" << std::endl;
    c = add_numbers(a, b);
    std::cout << "Statement3" << std::endl;
    std::cout << "Statement4" << std::endl;

    return 0;
}
```

Linking



```
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.8.0-pre.3.0
** Copyright (c) 2020 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86'

C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview>
C:\Program Files (x86)\Microsoft Visual Studio\2019\Preview>
```

Declarations and definitions

Declaration introduces the name in a file, a definition says what that name is or what it does.

One Definition Rule

Definitions can’t show up more than once in entire program, or Translation units.

Linkage

- No linkage
- Internal Linkage
- External Linkage
- Module Linkage(later)

Flipping Linkage

main.cpp

```
#include <iostream>

extern const double distance {45.9}; // Declaration & definition

void print_distance();// Declaration
void some_function(); // Declaration

int main(int argc, char **argv)
{
    std::cout << "distance(main) : " << distance << std::endl;
    std::cout << "&distance(main) : " << &distance << std::endl;

    std::cout << std::endl;
    print_distance();

    std::cout << std::endl;
    some_function();
    return 0;
}
```

some_other_file.cpp

```cpp
#include <iostream>

extern const double distance; // Declaring const double variable, defined in
                              // some_other_file

void print_distance(){
    //++distance; // Error : can't modify a read only variable.
    std::cout << "distance(some_other_file) : " << distance << std::endl;
    std::cout << "&distance(some_other_file) : " << &distance << std::endl;
}

/*
static void some_function(){
    std::cout << "some_function called ..." << std::endl;
}
*/

namespace{
    void some_function(){
        std::cout << "some_function called ..." << std::endl;
    }
}
```

Inline variables and functions

utility1.cpp

```
#include <iostream>

inline int age{12};

inline void some_function(){
    std::cout << "age : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}
```

utility2.cpp

```
#include <iostream>

inline int age{12};

inline void some_function(){
    std::cout << "age : " << age << std::endl;
    std::cout << "&age : " << &age << std::endl;
}
```

inline

Static(anonymous namespaces)

Forward declarations

```
class Dog; //Forward declaration

class Farm
{
public:
    Farm();
    ~Farm();

    void use_dog(const Dog& dog_param); // Doesn't require the definition
};
```

Slide intentionally left empty