

Programming with C++

COMP2011: Separate Compilation

Cecia Chan
Cindy Li
Brian Mak

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
Hong Kong SAR, China



Part I

Separate Compilation



Motivation Example: Mutual Recursion

```
#include <iostream>          /* File: odd-even.cpp */
using namespace std;

bool even(int);

bool odd(int x) { return (x == 0) ? false : even(x-1); }

bool even(int x) { return (x == 0) ? true : odd(x-1); }

int main()
{
    int x;
    cin >> x;                // Assume x > 0

    cout << boolalpha << odd(x) << endl;
    cout << boolalpha << even(x) << endl;

    return 0;
}
```

- The odd-even example consists of 3 functions:
 - `bool odd(int);`
 - `bool even(int);`
 - `int main();`
- Now instead of putting them all in **one** .cpp file, we would like to put **each** function in a **separate** .cpp file of its own.
- There are good reasons for doing that:
 - We can then easily **reuse** a function in another program.
 - In a big project, programmers work in a team. After the program framework is designed in terms of a set of **function prototypes**, each programmer writes **only some** functions.
 - If a function needs to be changed, **only one** file needs to be modified.
- But how to compile the **separate** files into **one single executable program**?

Solution #1: Separate Compilation

- In order that each file can be **separately compiled** on its own, each file must know the **existence** of every variable, constant, function that it uses.
- All **global** constants, variables, functions that are *used* in a file “A” but are defined in **another file** “B” must be **declared** in file “A” **before** they are used in the file.
 - **global constants**: repeat their definitions
 - **external variables**: add the keyword **extern**
 - **external functions**: add their function prototypes. The keyword **extern** is **optional** since all C++ functions are global anyway.
- The keyword **extern** in front of a variable/function means that the variable/function is **global** and is **defined** in **another** file.
- Usually put all **external declarations** at the **top** of a file. Why?

Solution #1: Separate Compilation — main()

```
#include <iostream>      /* File: main.cpp */
using namespace std;

/* Constant definitions */
const int MAX_CALLS = 100;

/* Global variable definition */
int num_calls;

/* Function declarations */
extern bool odd(int);    // "extern" is optional for functions

int main()
{
    int x;
    while (cin >> x)      // Assume x > 0
    {
        num_calls = 0; cout << boolalpha << odd(x) << endl;
    }

    return 0;
}
```

Solution #1: Separate Compilation — even()

```
#include <iostream> /* File: even.cpp */
#include <cstdlib>
using namespace std;

/* Constant definitions */
const int MAX_CALLS = 100;

/* Global variable declarations */
extern int num_calls;    // "extern" is a must for global variables

/* External function declarations */
extern bool odd(int);    // "extern" is optional for functions

bool even(int x)
{
    if (++num_calls > MAX_CALLS)
    {
        cout << "max #calls exceeded\n"; exit(-1);
    }

    return (x == 0) ? true : odd(x-1);
}
```

Solution #1: Separate Compilation — odd()

```
#include <iostream>      /* File: odd.cpp */
#include <cstdlib>

using namespace std;

/* Constants definitions */
const int MAX_CALLS = 200;

/* Global variable declarations */
extern int num_calls;    // "extern" is a must for global variables

/* Function declarations */
extern bool even(int);   // "extern" is optional for functions

bool odd(int x)
{
    if (++num_calls > MAX_CALLS)
    {
        cout << "max #calls exceeded\n"; exit(-1);
    }

    return (x == 0) ? false : even(x-1);
}
```


Solution #1: Separate Compilation Procedure 1

- Compile all the **source** .cpp files with the following command:

```
g++ -o odd-even main.cpp even.cpp odd.cpp
```

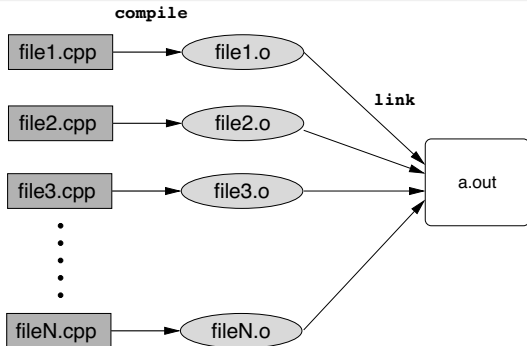
But this will again compile **all files** even if you may only change **one** of the file.

- Better to compile them **separately**:

```
g++ -c main.cpp  
g++ -c even.cpp  
g++ -c odd.cpp  
g++ -o odd-even main.o even.o odd.o
```

- The command `g++ -c a.cpp` will produce an **object** file “a.o” for the **source** file “a.cpp”.
- Then the final line `g++ -o odd-even main.o even.o odd.o` invokes the **linker** to link or merge the separate **object** files into one **single executable** program “odd-even”.

Solution #1: Separate Compilation Procedure 2



- Now, if you later modify only “main.cpp”, then you just need to **re-compile** “main.cpp” and **re-link** all **object .o** files.

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

```
g++ -o odd-even main.o even.o odd.o
```

- In general, just **re-compile** those source files that are modified, and **re-link** all object files of the project.

Part II

Definition vs. Declaration and Header Files

Variable and Function Definition

A **definition** introduces the **name** and **type** of an identifier such as a **variable** or a **function**.

- A **variable definition** requires the compiler to reserve an amount of **memory** for the variable as required by its **type**.
- A variable may also be initialized in its **definition**. For instance, `int x = 5;` .
- A **function definition** generates **machine codes** for the function as specified by its (function) **body**.
- In both cases, **definition** causes **memory** to be allocated to store the **variable** or **function**.
- A variable and function identifier must be defined exactly once in the **whole** program even if the program is written in separate files.

Variable and Function Declaration

The **declaration** of a **variable** or **function** announces that the variable or function **exists** and is **defined somewhere** — in the same file, or in a separate file.

- A variable's **declaration** consists of the its **name** and **type** preceded by the keyword **extern**. No initialization is allowed.
- A function's **declaration** consists of the its **prototype**, and may be **optionally** preceded by the keyword **extern**.
- A declaration does **not** generate codes for a **function**, and does **not** reserve memory for a **variable**.

Variable and Function Declaration ..

- There can be **many declarations** for a variable or function in the whole program.
- An identifier must be **defined** or **declared** before it can be used.
- During **separate compilation**, the compiler generates necessary information so that when the **linker** combines the **separate object files**, it can tell that the **variable/function declared** in a file is the same as the **global variable/function defined** in another file, and they should share the **same memory** or **codes**.

Header Files

- In Solution#1, you see that many global variable or function **declarations** are repeated in “odd.cpp” and “even.cpp”. That is undesirable because:
 - We are lazy, and we do not want to **repeat** writing the same **declarations** in multiple files.
 - Should a **declaration** require updating, one has to go through all files that have the declaration and make the change.
 - More importantly, maintaining **duplicate information** in multiple files is **error-prone**.
- The solution is to use **.h header** files which contains
 - **definitions** of global **variables** and **constants**
 - **declarations** of global **variables** and **functions**
- **Header files** are inserted to a file by the **preprocessor directive** **#include**.

```
#include <iostream> // standard library header files
#include "my_include.h" // user-defined header files
```

Solution #2: Separate Compilation — Header Files

```
/* File: my_include.h */
/* Include system or user-defined header files */
#include <iostream>
#include <cstdlib>
using namespace std;

/* Constant definitions */
const int MAX_CALLS = 100;

/* External function declarations */
extern bool odd(int); // "extern" is optional for functions
extern bool even(int);
```

```
/* File: global.h */
/* Global variable definitions */
int num_calls;
```

```
/* File: extern.h */
/* External global variable declarations */
extern int num_calls;
```


Solution #2: Separate Compilation — main()

```
#include "my_include.h" /* File: main.cpp */
#include "global.h"

int main()
{
    int x;
    while (cin >> x) // assume x > 0
    {
        num_calls = 0;
        cout << boolalpha << odd(x) << endl;
    }

    return 0;
}
```

Solution #2: Separate Compilation — even()

```
#include "my_include.h" /* File: even.cpp */
#include "extern.h"

bool even(int x)
{
    if (++num_calls > MAX_CALLS)
    {
        cout << "max #calls exceeded\n";
        exit(-1);
    }

    return (x == 0) ? true : odd(x-1);
}
```

Solution #2: Separate Compilation — odd()

```
#include "my_include.h" /* File: odd.cpp */
#include "extern.h"

bool odd(int x)
{
    if (++num_calls > MAX_CALLS)
    {
        cout << "max #calls exceeded\n";
        exit(-1);
    }

    return (x == 0) ? false : even(x-1);
}
```

Header Files of the Standard C++ Libraries

- **iostream**: input/output functions
- **iomanip**: input/output manipulation functions
- **cctype**: character functions
e.g. `int isdigit(char); int isspace(char); int isupper(char);`
- **cstring** C string functions:
e.g. `int strlen(const char []);`
`int strcmp(const char [], const char []);`
- **cmath**: math functions
e.g. `double sqrt(double); double cos(double);`
- **cstdlib**: commonly used functions
e.g. `int system(const char []); int atoi(const char []);`
`void exit(int); int rand(); void srand(unsigned int);`