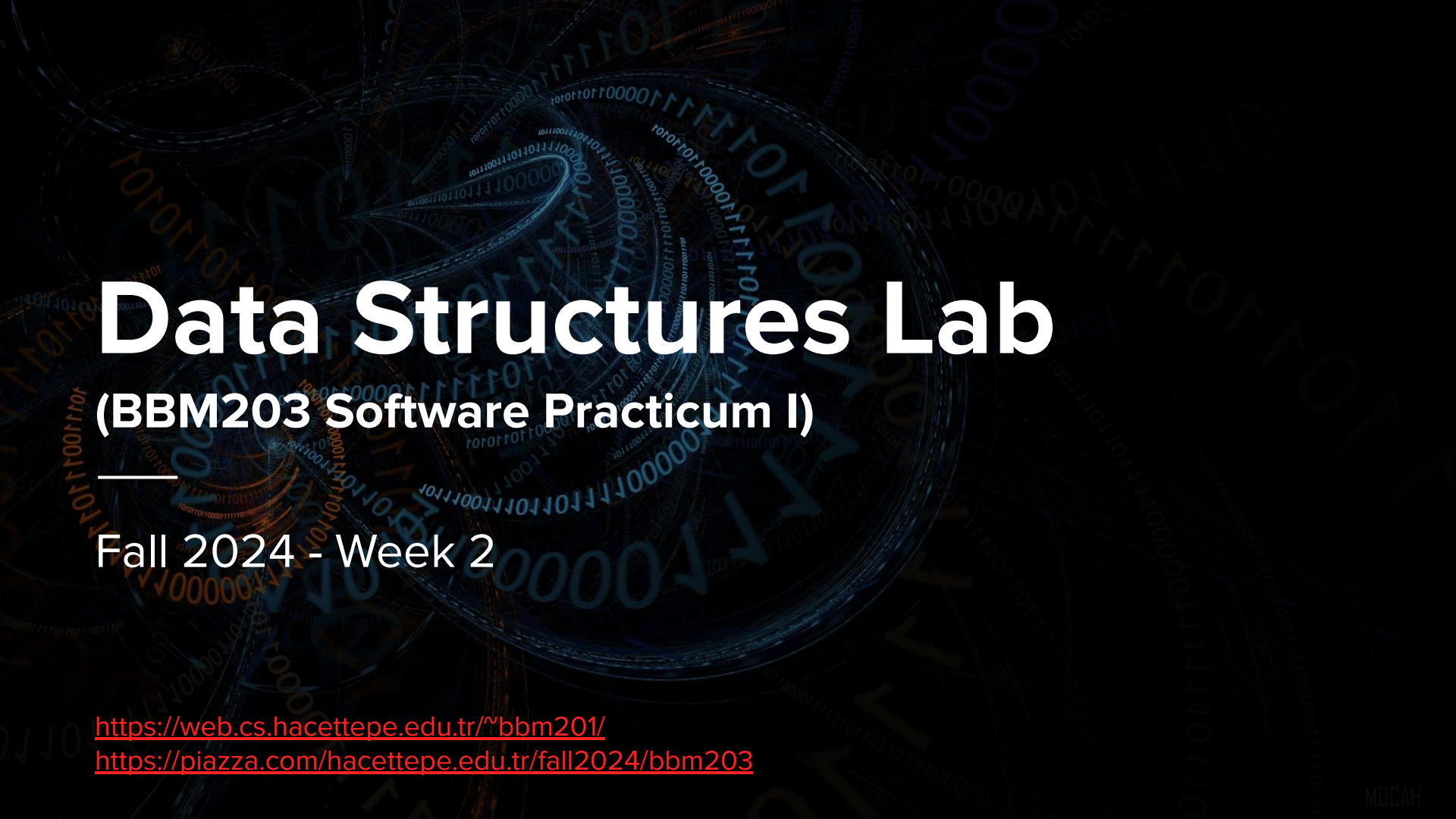


Data Structures Lab

(BBM203 Software Practicum I)

Fall 2024 - Week 2

https://web.cs.hacettepe.edu.tr/~bbm201/
https://piazza.com/hacettepe.edu.tr/fall2024/bbm203

Part 2: Java to C++ Transition Tutorial

Topics

- ➔ Debugging Tips
- ➔ Makefiles

Debugging

Errors in C++

1. Compile Time Errors

Compile-time errors in C++ occur during the compilation phase and prevent the successful generation of the executable binary. The common types are:

- **Syntax Errors**
- **Type Errors**
- **Name Errors**
- **Undefined Symbols**
- **Redeclaration Errors**

```
void main()
{
    int x = 10;
    int y = 15;

    cout << " "<< (x, y) // semicolon missed
}
```

Error messages after trying to compile:

```
main.cpp:4:1: error: ‘::main’ must return ‘int’
    4 | void main()
      | ^~~~
main.cpp: In function ‘int main()’:
main.cpp:9:25: error: expected ‘;’ before ‘}’ token
    9 |     cout << " "<< (x, y) // semicolon missed
      |                         ^
      |                         ;
   10 | }
      | ~
```

Errors in C++

2. Run-time Errors

Happen during program execution (run-time) after successful compilation.

Memory Access Violation:
This occurs when a program tries to access a memory location that it is not allowed to access (e.g., accessing an array out of bounds).

Java programmer writing C++


```
int arr[5];
arr[10] = arr[12] + arr[13];  // Causes a segmentation fault
```

> Segmentation fault

Errors in C++

2. Run-time Errors

Happen during program execution (run-time) after successful compilation.

Floating Point Exception:

This occurs when you attempt to divide by zero or perform an invalid floating-point operation.

```
int a = 10;
int b = 0;
int c = a / b;
```

> Floating point exception

Errors in C++

2. Run-time Errors

Happen during program execution (run-time) after successful compilation.

Null Pointer Dereference:

This happens when you attempt to access a member or data through a null pointer.

```
int *var = nullptr; //pointer of type integer that stores "nullptr"
cout << "var -> ";
cout << *var;
cout << "This will not print";
```

> Segmentation fault

Errors in C++

2. Run-time Errors

Happen during program execution(run-time) after successful compilation.

Uninitialized Variable Usage:
Using a variable without initializing it can lead to unpredictable behavior and runtime errors.

```cpp
int i; // uninitialized variable

// WARNING: This causes undefined behaviour!
std::cout << i << '\n';
```

Errors in C++

2. Run-time Errors

Happen during program execution (run-time) after successful compilation.

Logic Errors (Bugs):
These are errors that occur due to incorrect algorithm or program design, resulting in unexpected outcomes.

The if statement should check “if the command is not SEND **and** the command is not RECEIVE”, instead it checks “if the command is not SEND **or** the command is not RECEIVE”. Therefore, if it encounters SEND command, for example, it will mistakenly execute the statements inside the block.

```
if (command[0].compare("SEND") != 0 || command[0].compare("RECEIVE") != 0) {
    // Do something
}
```

Errors in C++

3. Linker Errors

Linker errors in C++ occur during the linking phase of the compilation process. The linker is responsible for combining object files and libraries into an executable program. Linker errors typically result from unresolved symbols, duplicate symbols, or incorrect usage of libraries.

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

void Main() // Here Main() should be main()
{
    int a = 10;
    cout << " "<< a;
}
```

```
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/9/../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x24): undefined reference to `main'
collect2: error: ld returned 1 exit status
-bash: ./main: No such file or directory
```

Memory Leaks

Memory leaks in C++ occur when a program fails to release memory that was previously allocated dynamically, leading to a gradual consumption of system resources. These leaks can lead to performance issues and ultimately affect the program's stability.

In Java, memory management is handled automatically by the JVM (Java Virtual Machine) through a process called garbage collection, **unlike C++**.

```
examples $g++ main.cpp -o main -g
examples $valgrind ./main
==370== Memcheck, a memory error detector
==370== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==370== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==370== Command: ./main
==370==
==370==
==370== HEAP SUMMARY:
==370==     in use at exit: 16 bytes in 1 blocks
==370==   total heap usage: 2 allocs, 1 frees, 72,720 bytes allocated
==370==
==370== LEAK SUMMARY:
==370==    definitely lost: 16 bytes in 1 blocks
==370==    indirectly lost: 0 bytes in 0 blocks
==370==      possibly lost: 0 bytes in 0 blocks
==370==    still reachable: 0 bytes in 0 blocks
==370==         suppressed: 0 bytes in 0 blocks
==370== Rerun with --leak-check=full to see details of leaked memory
==370==
==370== For lists of detected and suppressed errors, rerun with: -s
==370== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Debugging in C++

- Debugging in C++ involves using gdb (either through an IDE) or valgrind.
- **gdb** allows for setting breakpoints and viewing the state of your program.
- **valgrind** checks for memory leaks and memory errors.
- Debugging is done in order to fix logic errors or runtime errors.
- IDEs like CLion, DevC++ and codeblocks supply quite explanatory visual debugging functionality.

Makefiles

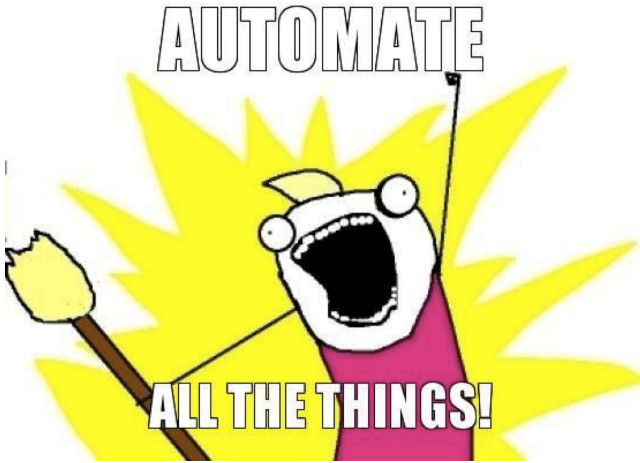
The Makefile Utility - Why Makefiles?

- Makefiles are a simple way to organize code compilation.
- Lots of source files:

  `foo1.h, foo2.h, foo1.cpp, foo2.cpp, main.cpp`, etc.
- How to manage them all?
- Compiling is complicated.

  Solution: Makefile

- Make - automatically build and manage your programs.
- Compile quickly with a single command.
- Recompiling is even quicker.



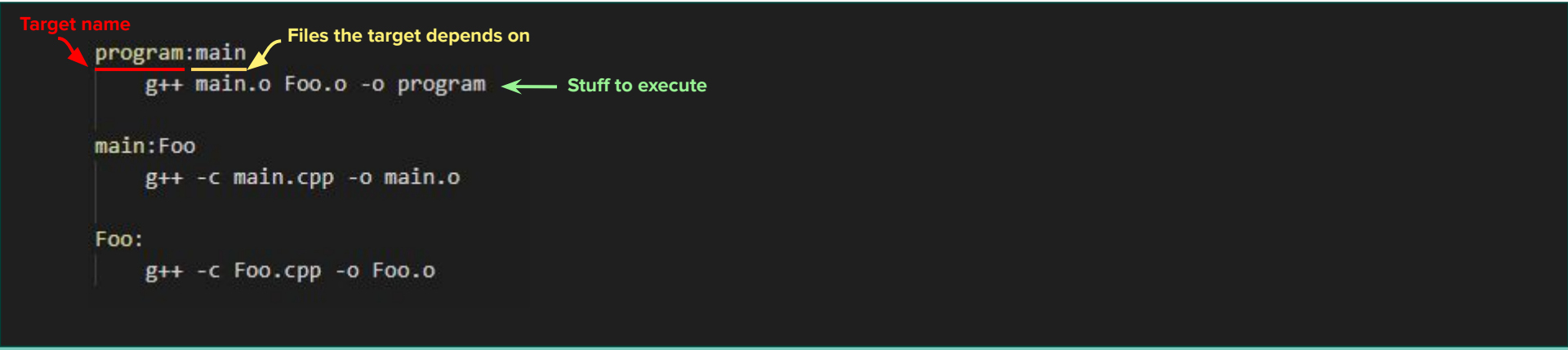
Makefiles

Makefile Syntax:

A Makefile consists of a set of *rules*. A rule generally looks like this:

```
targets: prerequisites
  command
  command
  command
```

A sample Makefile (saved under the filename **`Makefile`** without any extensions!):



Makefiles

Compile and Run the ‘Traditional’ Way:

```
g++ main.cpp Foo1.cpp Foo2.cpp -o program
./program
```

Think of having tens or hundreds of files to compile each time.

Compile and Run with a Makefile:

```
make
./program
```

Compile all files with one word: make

For more information on Makefiles you can check: https://makefiletutorial.com/

Makefiles Exercise: Test Yourself Now

Take the practice example from the last week and modify it such that the summing of the numbers from 1 to n (some positive integer n) is now placed in a separate function named **`sum_numbers(int n)`** that is defined in another .cpp file named **`helper_functions.cpp`**. Modify the **`main.cpp`** such that it includes only the function definition of **`sum_numbers(int n)`**(no implementation!) and calls this function to calculate the sum. Write a **`Makefile`** that will compile all .cpp functions and produce an executable named **`my_sum_program`**.

When you execute the following commands:

```
make
./my_sum_program
```

Your program should perform the same functionality as your main.cpp from the first exercise: print the result in the following format:

```
The sum is: X
```

More Useful Resources For Practice:

- https://www.w3resource.com/cpp-exercises/basic/index.php
- https://www.w3schools.com/cpp/cpp_exercises.asp
- https://www.hackerrank.com/domains/cpp
- https://algoleague.com/

Questions?