OOP Lab1

Nica David Andrei

TO-DO’s

1. How many lines of code a popular video game has?

Age of Empires online – 1 million lines of code

1. How would I manage this amount of code?

Creation of the best relevant and important functions and then call them to see if they can be used again for a more time and resources efficient code.

*1.3*

1. What compile / link / build / run means?

- **Compile**: This is the process of converting source code written in a high-level programming language (like C++ or Java) into machine code (binary code) that a computer’s processor can execute. The result of this process is usually an object file.

- **Link**: Linking is the process of combining multiple object files into a single executable or library. This step resolves references between different parts of the program, such as function calls and variable accesses that are spread across multiple files.

- **Build**: Building is a broader term that encompasses both compiling and linking, along with other steps like preprocessing and assembling. Essentially, it’s the entire process of converting source code into a final executable program.

- **Run**: Running is the execution of the compiled and linked program. This is when the program is actually loaded into memory and executed by the computer’s processor.

1. What release / debug means?

**Debug**:

Purpose: Used during development to test and debug the program.

Features: Includes additional debug information, such as symbols and line numbers, which help developers trace and fix issues. It also disables optimizations to make debugging easier.

Performance: Generally slower and larger in size due to the extra debug information and lack of optimizations.

**Release**:

Purpose: Used for the final version of the program that is distributed to users.

Features: Optimized for performance and size, with debug information removed or minimized. This makes the program run faster and take up less space.

Performance: Faster and more efficient, but harder to debug if issues arise because the debug information is not included.

1. What is an IDE?

An IDE (Integrated Development Environment) is a software application that provides comprehensive facilities to programmers for software development. Popular examples of IDEs include Visual Studio, IntelliJ IDEA, Eclipse, and PyCharm.

Key components/features

* **Source Code Editor**: This is where you write your code. It often includes features like syntax highlighting, code completion, and error detection to make coding easier and more efficient.
* **Build Automation Tools**: These tools help automate the process of compiling and linking your code, making it easier to build your project with a single command or click.
* **Debugger**: A debugger helps you find and fix errors in your code by allowing you to run your program step-by-step, inspect variables, and set breakpoints.
* **Version Control Integration**: Many IDEs integrate with version control systems like Git, allowing you to manage changes to your codebase directly within the IDE.
* **Other Tools**: Some IDEs include additional tools like GUI designers, database management tools, and more, depending on the programming language and the specific needs of the developer123.

1. Use couple of IDE’s to write C code. What are the differences ? Why do you use your preferred IDE?

Code Blocks

- Type: Desktop IDE

- Features:

* Open-source and free.
* Supports multiple compilers like GCC, Clang, and Visual C++.
* Highly customizable with numerous plugins.
* Integrated debugger with full breakpoint support.
* Cross-platform (Windows, Linux, macOS).

- Pros:

* Lightweight and fast.
* Easy to set up and use.
* Good for beginners due to its simplicity.

- Cons:

* Limited advanced features compared to some other IDEs.
* The interface might feel outdated to some users.

Visual Studio Code

- Type: Desktop Code Editor

- Features:

* Open-source and free.
* Extensive customization through a rich library of extensions.
* Powerful features like IntelliSense, debugging, and Git integration.
* Supports a wide range of programming languages.
* Cross-platform (Windows, Linux, macOS).

- Pros:

* Highly customizable and extensible.
* Strong community support and regular updates.
* Suitable for both small and large projects.

- Cons:

* Requires setup and customization to fully leverage its capabilities.
* Can be resource-intensive with many extensions.

Replit

- Type: Browser-based IDE

- Features:

* Web-based, no installation required.
* Supports over 50 programming languages.
* Real-time collaboration and interactive tutorials.
* Integrated terminal and database.
* Accessible from any device with an internet connection.

- Pros:

* User-friendly and minimal setup.
* Excellent for collaborative projects and educational use.
* Accessible from anywhere, making it convenient for coding on the go.

- Cons:

* Limited customization compared to desktop IDEs.
* Performance can be less efficient for very large projects.
* Some advanced features may require a paid plan¹²³.

I personally prefer to use Visual Studio Code since its interface and options are much better than that of Code Blocks and I prefer to have the ability to access things regardless of my current connection to the internet, something which you can’t really do with Replit.

1. Investigate if there is an IDE which suggests the function parameters, even if the function is included in libraries. Is it possible to get information about how to use function ?

Some of the IDE’s that suggest the function parameters are:

* Intellij IDEA
* Visual Studio Code
* Clion
* Eclipse CDT

Most modern IDEs provide ways to get detailed information about functions, including:

* **Hover Documentation**: Hovering over a function name often displays a tooltip with documentation and parameter details.
* **Quick Documentation**: Many IDEs have a shortcut (like Ctrl+Q in IntelliJ IDEA) to bring up detailed documentation for the selected function.
* **Integrated Documentation Browsers**: Some IDEs integrate with documentation browsers or online resources to provide comprehensive information about functions and libraries.

1. Investigate naming conventions

Naming conventions in programming are standardized guidelines for naming variables, functions, classes, and other entities in code. These conventions help ensure consistency, readability, and maintainability of code, making it easier for teams to collaborate and understand each other’s work. Some examples are:

* Camel Case (typically used for variable and function names)
* Snake Case (used for variable names in languages like Python)
* Kebab Case (commonly used in URLs and CSS class names)

Homework

T.1.5. (Optional) Investigate other libraries in Windows programming: header / .lib / .dll What are the differences ? Basic example for each one.

In Windows programming, libraries can come in different forms: header files, static libraries (.lib), and dynamic-link libraries (.dll). Here we have examples for each:

1. Header Files (.h) Description: • Header files typically contain declarations of functions, classes, constants, and macros. They are not compiled on their own but are included in other source files to provide necessary declarations.

// func\_opt3.h: #ifndef FUNC\_OPT3\_H #define FUNC\_OPT3\_H void sayHello(); #endif

//main.cpp : #include #include "func\_opt3.h"

void sayHello() {std::cout << "Hello from header!" << std::endl;}

int main() { sayHello(); return 0; }

1. Static Libraries (.lib) Description: • A static library is a collection of object files that are linked into the executable at compile time. Once linked, the library becomes part of the executable, which increases its size but improves performance as there are no runtime dependencies. Create a static library: // static\_opt3.cpp #include

void sayHello() { std::cout << "Hello from static library!" << std::endl; }

Use the static library: // main.cpp: #include #include "my\_functions.h" extern "C" void sayHello(); // Declaration of the function from the static library

int main() { sayHello(); return 0; }

1. Dynamic-Link Libraries (.dll) Description: • A DLL is a dynamic library that is loaded at runtime. This allows for smaller executable sizes and the ability to share code among multiple applications. However, it introduces runtime dependencies, meaning the application must be able to locate the DLL when it runs. Create a DLL: // dll\_opt3.cpp #include #include <windows.h>

extern "C" \_\_declspec(dllexport) void sayHello() { std::cout << "Hello from DLL!" << std::endl; }

// main.cpp: #include #include <windows.h>

typedef void (\*HelloFunc)();

int main() { HINSTANCE hinstLib = LoadLibrary(TEXT("my\_dll.dll")); if (hinstLib != NULL) { HelloFunc sayHello = (HelloFunc)GetProcAddress(hinstLib, "sayHello"); if (sayHello != NULL) { sayHello(); // Call the function from the DLL } FreeLibrary(hinstLib); } else { std::cout << "Failed to load DLL!" << std::endl; } return 0; }

Summary • Header Files (.h): o Contain declarations, no compiled code. o Must be included in source files. • Static Libraries (.lib): o Contain compiled code that is linked at compile time. o No runtime dependencies. o Increases the size of the executable. • Dynamic-Link Libraries (.dll): o Contain compiled code that is linked at runtime. o Smaller executables, can share libraries across applications. o Must be available at runtime.