Week#15

Programming Practicals (EXTRAS)

Outline

1. File organisation
2. Compiling mechanism
3. Performance measurement

File Organisation

File Organisation – Why?

Why Organize Code into .h and .c Files?

1. **Problem Solving Mindset**: Good programmers don't just write working code — they **structure** code for clarity, reuse, and collaboration.
2. **Separation of concerns**: Interface (.h) vs Implementation (.c)
3. **Modularization**:
   - Easier debugging and unit testing
   - Each part solves a subproblem, aligned with decomposition
4. **Reusability**: A well-structured library can be reused across projects
5. **Teamwork-Friendly**: Teams can work on different modules concurrently

File Organisation – What?

What Is a C Library?

1. A **C library** is a set of functions and definitions grouped together for reuse.
2. A library typically consists of:
   - **Header file (.h)** → Function declarations, constants, macros
   - **Source file (.c)** → Function definitions (code)
3. **Not a library**:
   - A single .c file with everything
   - A .h file that includes full function definitions

File Organisation – What?

What the structure of C library? **mylib.h – Header File (Interface)**

```
// Header guard to prevent duplicate inclusion
#ifndef MYLIB_H
#define MYLIB_H

// Function prototypes (declarations)
int add(int a, int b);
int multiply(int a, int b);

#endif
```

File Organisation – What?

What the structure of C library? **mylib.h – Source File (Implementation)**

```
#include "mylib.h"

// Function definitions
int add(int a, int b) {
    return a + b;
}

int multiply(int a, int b) {
    return a * b;
}
```

File Organisation – What?

What the structure of C library? **main.c – Application Entry Point**

```
#include <stdio.h> // Standard library
#include "mylib.h” // Our own library

// Main function
int main() {
    printf("Sum: %d\n", add(3, 5));
    printf("Product: %d\n", multiply(3, 5));
    return 0;
}
```

File Organisation – How?

Best Practices in File Organization

1. Separate interface (.h) and implementation (.c)
2. Avoid putting actual code in .h files
3. Use header guards (`#ifndef, #define or #endif`)
4. Group related functions together in libraries
5. Keep main.c clean and high-level

Compiling Mechanism

Compiling Mechanism – Why?

Why need compilation?

1. **Compilation = Building**: Your .c files must be translated into machine code.
2. **Multiple files = more control**: As your program grows, you need to manage separate files and their dependencies.
3. **Makefiles = automation**: No need to manually compile every file every time.

Compiling Mechanism – How?

How to compile?

```
gcc main.c mylib.c -o myprogram
./myprogram
```

Explanation:

1. `gcc` = GNU compiler
2. `main.c + mylib.c` → compiled and linked
3. `-o myprogram` = output file name
4. `./myprogram` = run the compiled program

Compiling Mechanism – Why?

Why compiling with makefile?

1. Avoid repeating long gcc commands
2. Only recompile what has changed
3. Organise bigger projects

Compiling Mechanism – How?

How to compile with makefile?

```
CC = gcc
CFLAGS = -Wall
OBJ = main.o mylib.o

myprogram: $(OBJ)
	$(CC) -o $@ $^

main.o: main.c mylib.h
	$(CC) -c main.c $(CFLAGS)

mylib.o: mylib.c mylib.h
	$(CC) -c mylib.c $(CFLAGS)

clean:
	rm -f *.o myprogram
```

Compiling Mechanism – How?

How to compile with makefile?

```
CC = gcc
CFLAGS = -Wall
OBJ = main.o mylib.o

myprogram: $(OBJ)
	$(CC) -o $@ $^
```

CC = gcc → sets the compiler

CFLAGS = -Wall → enable warnings

OBJ = main.o mylib.o → list object files

myprogram: $(OBJ) → Rule: build myprogram using main.o and mylib.o

`$(CC) -o $@ $^` → Compile and link. $@ = target (myprogram), $^ = dependencies (main.o mylib.o)

Compiling Mechanism – How?

How to compile with makefile?

```
main.o: main.c mylib.h
	$(CC) -c main.c $(CFLAGS)

mylib.o: mylib.c mylib.h
	$(CC) -c mylib.c $(CFLAGS)

clean:
	rm -f *.o myprogram
```

main.o rules → Rule: if main.c or mylib.h changes, recompile main.o; Compile main.c to main.o only

mylib.o rules → Rule: recompile mylib.o only if needed

clean rule → Custom target to delete compiled files

`rm -f *.o myprogram` → Deletes object files and the binary

Compiling Mechanism – How?

How to compile with makefile?

```
make         # builds the program
./myprogram # runs it
make clean  # removes all generated files
```

Performance Measurement

Performance Measurement – Why?

Why need to measure the performance?

1. Problem-Solving = Efficient Solutions, not just correctness
2. Real-World Constraints: time and memory efficiency
3. Detect inefficiencies like slow loops or memory leaks
4. Enable iterative optimization (Measure → Improve)

Performance Measurement – Why?

Why need to measure the performance?

Many applications are resource-critical:

Application	Time critical?	Space critical?
Real-time drone	Yes	Yes
Image processing	Yes	Yes
Embedded systems	Maybe	Yes
Web services	Yes	Maybe

Performance Measurement – What?

What do we measure?

1. Execution Time: how long code takes to run
2. Memory Usage: how much RAM is used
3. CPU Usage: how much processing power is used
4. I/O Performance: speed of file or network operations

Focus Today: Time and Space

Performance Measurement – What?

What tools do we have to measure time consumption?

```
// Using clock()
#include <time.h>
clock_t start = clock();
// code block
clock_t end = clock();
double elapsed = (double)(end - start) / CLOCKS_PER_SEC;

// Using gettimeofday() (Linux/UNIX)
#include <sys/time.h>
struct timeval start, end;
gettimeofday(&start, NULL);
// code block
gettimeofday(&end, NULL);
double elapsed = (end.tv_sec - start.tv_sec) +
                 (end.tv_usec - start.tv_usec) / 1e6;
```

Performance Measurement – How?

Example of time measurement (using clock())

```
#include <stdio.h>
#include <time.h>

int main() {
    clock_t start = clock();

    for (volatile int i = 0; i < 100000000; i++);

    clock_t end = clock();
    double time_spent = (double)(end - start) / CLOCKS_PER_SEC;

    printf("Time elapsed: %f seconds\n", time_spent);
    return 0;
}
```

Performance Measurement – How?

Example of time measurement (using gettimeofday())

```
#include <stdio.h>
#include <sys/time.h>

int main() {
    struct timeval start, end;

    gettimeofday(&start, NULL);
    volatile long long sum = 0;
   for (long long i = 0; i < 100000000; i++);
    gettimeofday(&end, NULL);

    double elapsed = (end.tv_sec - start.tv_sec) + (end.tv_usec - start.tv_usec) / 1e6;
   printf("Elapsed time: %f seconds\n", elapsed);
   return 0;
}
```

Performance Measurement – How?

Example of resource measurement

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/resource.h>

int main() {
    int *arr = malloc(1000000 * sizeof(int));
    struct rusage usage; // Structure to hold resource usage info
    getrusage(RUSAGE_SELF, &usage); // Get resource usage for the current process
    printf("Memory used: %ld KB\n", usage.ru_maxrss); // Print the memory used in KB
    free(arr);
    return 0;
}
```

Exercise

Organise any of the previous codes you have written in this course into a library structure, compute the time consumption to run it.

Thank you! See you in another course maybe?