Systems Programming - Labwork 10

Focus: C preprocessor basics, `gcc -E`, and Makefile structure. Work in a scratch directory (e.g., `~/labwork10`).

C Preprocessor

Do not include `<stdio.h>`. Use only the preprocessor constructs requested. Submit `expanded.c` and your explanation.

Track A

1. Header file `myhdr.h`: declare `b` as integer with value `3`.
2. Main program `myprg.c` using preprocessor directives:
   - Include `myhdr.h`.
   - Define `A` with value `4`.
   - Define a function-like macro `f(x,y)` that calculates `(x+y)*(A+5)`.
   - In `main`, declare variables `a` and `c` with values `6` and `0` respectively.
   - Then calculate `c = f(a,b) - 1;`
3. Preprocess only (no compile):

```
gcc -E myprg.c -o expanded.c
```

4. Observe substitutions in `expanded.c` and explain:
   - What changed after preprocessing?
   - Does the C preprocessor do any calculations? Why?

Track B

1. Header file `myhdr.h`: declare `b` as integer with value `2`.
2. Main program `myprg.c` using preprocessor directives:
   - Include `myhdr.h`.
   - Define `A` with value `3`.
   - Define a function-like macro `f(x,y)` that calculates `(x+y)*(A+1)`.
   - In `main`, declare variables `a` and `c` with values `5` and `0` respectively.
   - Then calculate `c = f(a,b) / 2;`
3. Preprocess only (no compile):

```
gcc -E myprg.c -o expanded.c
```

4. Observe substitutions in `expanded.c` and explain:
   - What changed after preprocessing?
   - Does the C preprocessor do any calculations? Why?

Makefile

Write a Makefile for a small C program (same `myprg.c` from Q1). Do not paste a full working Makefile from the internet. Your submission must show that you understand how each part works.

Required elements

1. Variables for:
    - compiler (`CC`)
    - compiler flags (`CFLAGS`, include `-Wall -Wextra -std=c11`)
    - binary name (`BIN`)
2. A default target `all` that builds the binary.
3. A target for the binary that depends on `myprg.c`.
4. A `clean` target that removes the binary.
5. Mark `clean` as `.PHONY`.

Explain in your own words

Write 4-6 sentences explaining:

- What a target, prerequisite, and recipe are.
- Why variables are used in Makefiles.
- What `.PHONY` does and why `clean` should be phony.
- What happens when you run `make` with no arguments.

Hint (skeleton only)

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c11
BIN = myprg

all: $(BIN)

$(BIN): myprg.c
    $(CC) $(CFLAGS) -o $(BIN) myprg.c

clean:
    rm -f $(BIN)

.PHONY: all clean
```

Combined Challenge

Create a Makefile target named `preprocess` that runs `gcc -E` on `myprg.c` and writes `expanded.c`. Add a second target named `show` that prints the first 20 lines of `expanded.c`.

Explain why adding these targets is useful when studying the C preprocessor.

Example: Macro vs Function

Create two small programs to compare a macro call with a real function call. Use `clock()` to measure runtime and use `gcc -E` to inspect macro expansion.

Files to create

`macro_test.c`

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

#define SCALE 3
#define MUL(a,b) ((a) * (b) * SCALE)

int main(void)
{
    const int loops = 200000000;
    int sum = 0;
    clock_t start = clock();
    for (int i = 0; i < loops; i++) {
        sum += MUL(i, 2);
    }
    clock_t end = clock();
    printf("macro sum=%d time=%f\n", sum, (double)(end - start) /
CLOCKS_PER_SEC);
    return 0;
}
```

`func_test.c`

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

#define SCALE 3

static inline int mul(int a, int b)
{
    return a * b * SCALE;
}

int main(void)
{
    const int loops = 200000000;
    int sum = 0;
    clock_t start = clock();
    for (int i = 0; i < loops; i++) {
        sum += mul(i, 2);
    }
```

```
    clock_t end = clock();
    printf("func sum=%d time=%f\n", sum, (double)(end - start) /
CLOCKS_PER_SEC);
    return 0;
}
```

Build and inspect

```
gcc -O2 macro_test.c -o macro_test
gcc -O2 func_test.c -o func_test
./macro_test
./func_test

gcc -E macro_test.c -o macro_expanded.c
gcc -E func_test.c -o func_expanded.c
```

What to report

- Compare the runtime outputs and note any differences.
- Show the key expanded lines where `MUL(i, 2)` is replaced by an expression, while `mul(i, 2)` remains a function call.
- Explain why macros are text substitution and why functions may or may not be inlined depending on compiler optimization.

Submission

- Submit `myhdr.h`, `myprg.c`, `expanded.c`, and your written explanations.
- Submit the Makefile and Q2 explanation.
- Submit via YULEARN.