C PROGRAMMING
Lecture 11

1st semester 2023 - 2024

Macros

- A macro is a symbol that is recognized by the preprocessor and replaced by the macro body
- A preprocessor macro is used for:
  - Defining a symbolic name for a constant value
  - Defining an alias for something else
- To define a macro, use this directive:

  #define mname mtext

  mname → macro name, formal parameters allowed

  mtext → substitution text
- Examples:

```
#define   BUFFERSZ    1024
#define   WORDLEN   64
```

- The replacement is not done if mname appears within a character string or is part of a longer identifier

Macros

- The preprocessor expands macros in the C code
- Wherever it sees ***mname*** in the source file, it substitutes ***mtext*** in its place
- A macro definition can contain previously defined macros
- Macros are not the same as C variables! No storage is created in memory; macros only exist at compile-time
- An old C programming convention rules that macro names be fully capitalized to differentiate them from C variable names; this is NOT a language rule, hence NOT enforced by the C compiler!

Macros - Example

```
/* EUR rate in RON */
/* Working hours per week */
#include <stdio.h>
#define RATE1 4.485
#define HRS_WK1 40

int main (void)
{
   float rate2 = 4.50;
   float hrs_wk2 = 35;
   float pay1, pay2;

   /* Weekly pay in RON */
   pay1 = RATE1 * HRS_WK1;
   pay2 = rate2 * hrs_wk2;

   printf("RATE1=%f\n",RATE1);
   return 0;
}
```

Macros replacement - Example

- Replacement text can be **any** set of characters, meaning zero or more
- Example:

```
#define then
```

causes all occurrences of then to be removed from the source file.

```
if(a>b) then
max=a;
else
max=b;
```

```
if (a > b)
    max = a;
else
    max = b;
```

Parametrized Macros

- Macros can have parameters
  - these resemble functions in some ways:
    - macro definition ~ formal parameters
    - macro use ~ actual arguments
  - Form:

    #define macroName(arg$_1$, …, arg$_n$)
    ** replacement_list**

    IMPORTANT: no space between macro name and (
  - Example:

```
#define  deref(ptr)        *ptr
#define  max(x,y)          x > y ? x : y
```

Parametrized Macros

```
#include <stdio.h>

// Macro definition with parameters
#define ADD_TWO(x) ((x) + 2)

int main(void) {
    int num = 5;

    // Invoking the macro with the argument 5
    int result = ADD_TWO(num);

    printf("Result: %d\n", result);

    return 0;
}
```

Parametrized Macros

```
#include <stdio.h>

#define PRINT_MESSAGE(msg) printf("Message: %s\n", msg)

int main(void) {
    PRINT_MESSAGE("Hello, Macros!");
    return 0;
}
```

Parametrized Macros

```
#include <stdio.h>

// Macro definition to find the maximum of two values
#define MAX(x, y) ((x) > (y) ? (x) : (y))

int main(void) {
    int a = 10;
    int b = 7;

    // Using the MAX macro to find the maximum between a and b
    int max_value = MAX(a, b);

    printf("The maximum value between %d and %d is: %d\n", a, b, max_value);

    return 0;
}
```

Parametrized Macros

```
#include <stdio.h>

// Macro definition for a repetitive loop
#define REPEAT(n) \
    for (int i = 0; i < (n); ++i)

int main(void) {
    // Using the REPEAT macro to create a loop that prints numbers from 0 to 4
    REPEAT(5) {
        printf("%d ", i);
    }

    printf("\n");

    return 0;
}
```

Parametrized Macros

```
#include <stdio.h>

// Macro definition to find the maximum of two values using if-else
#define MAX(x, y) \
    do { \
        if ((x) > (y)) { \
            (x); \
        } else { \
            (y); \
        } \
    } while (0)

int main(void) {
    int a = 10;
    int b = 7;

    // Using the MAX macro to find the maximum between a and b
    int max_value = MAX(a, b);

    printf("The maximum value between %d and %d is: %d\n", a, b, max_value);

    return 0;
}
```

Parametrized Macros

```
#include <stdio.h>

// Macro definition to square a number
#define SQUARE(x) ((x) * (x))

// Macro definition to find the maximum of two values using SQUARE macro
#define MAX_SQUARE(a, b) (SQUARE(a) > SQUARE(b) ? SQUARE(a) : SQUARE(b))

int main(void) {
    int num1 = 3;
    int num2 = 5;

    // Using the MAX_SQUARE macro, which calls the SQUARE macro
    int max_square_value = MAX_SQUARE(num1, num2);

    printf("The maximum square value between %d and %d is: %d\n", num1, num2,
max_square_value);

    return 0;
}
```

Macros or Functions?

- Macros may be (sometimes) faster
  - don’t incur the overhead of function call/return
  - however, the resulting code size is usually larger
    - this can lead to loss of speed
- Macros are “generic”
  - parameters don’t have any associated type
  - arguments are not type-checked
- Macros may evaluate their arguments more than once
  - a function argument is only evaluated once per call

Macros or Functions?

- Macros and functions may behave differently if an argument is referenced multiple times:
  - a function argument is evaluated once, before the call
  - a macro argument is evaluated each time it is encountered in the macro body.

Macro Properties

- Macros may be nested
  - in definitions, e.g.:

```
#define  Pi          3.1416
#define  Twice_Pi    2*Pi
```

  - in uses, e.g.:

```
#define  double(x)    x+x
#define  Pi  3.1416
...
if ( x > double(Pi) ) ...
```

- Nested macros are expanded recursively

Macros - Pitfalls

```
#define MAX 10
#define LOWERMAX MAX – 1
maxvalue=LOWERMAX*5
```

What is the value of maxvalue?

How can this be avoided?

Macros - Pitfalls

```
#define MAX 10
#define LOWERMAX MAX – 1
maxvalue=LOWERMAX*5
```

What is the value of maxvalue?

How can this be avoided?

```
#define MAX 10
#define LOWERMAX (MAX - 1)
maxvalue = LOWERMAX * 5;
```

Macros - Pitfalls

```
#define square(n) n*n
y=square(10);
```

Result?

Now, consider

```
x=10;
y=square(x+1);
```

How can this be fixed?

Macros - Pitfalls

```
#define square(n) n * n

// First case
y = square(10);
// This gets expanded to y = 10 * 10, so y = 100

// Second case
x = 10;
y = square(x + 1);
// This gets expanded to y = x + 1 * x + 1
// Due to operator precedence, it becomes y = 10 + 1 * 10 + 1
// Resulting in y = 21
```

Macros - Pitfalls

```
#define square(n) ((n) * (n))

// Now, using the corrected macro
x = 10;
y = square(x + 1);
// This gets expanded to y = (x + 1) * (x + 1)
// Now, due to parentheses, it becomes y = (10 + 1) * (10 + 1)
// Resulting in y = 11 * 11 = 121
```

Macros - Pitfalls

Same macro as on previous slide

y=square(10)/square(5)

Solution?