Seminar 10: Preprocessing

Macros in C (15 minutes)

In C, the macro processor is quite simple compared to `nasm`. We remind you that using the -E switch in gcc / clang, you can see the result of preprocessing a file with the source code without starting its compilation, like `gcc -E file.c`.

You have several text replacement constructs at your disposal:

```
// text replacement
#define x 42
// text replacement with parameters
#define add_42(y) y + 42
```

Question 1 What will be output to `stdout` for the following piece of code? If this is an unexpected result, how do you rewrite the `dbl` to get rid of the unwanted and unexpected behavior?

```
#define dbl(y) y * 2

...
printf("%d", dbl(3+3));
```

There is also a set for organizing conditions:

```
#ifdef X
// this text will be included if X is defined with #define
#endif

#ifndef X
// this text will be included if X was not defined with #define
#endif
```

Converting to strings

Interesting is the ability to frame parameters with quotes using #.

```
#define print_var(x) printf(#x " is %d", x );
```

Question Write a program that will use this macro to output the variable. Test it with the `-E` switch for `gcc/clang`.

```
int main(void)
{
    int x = 23;
.
.
.
}
```

Question 2 What happens if you write `print_var (42)`?

Concatenation of strings

It is also possible to glue together identifiers from several parts with ##:

```
#include <inttypes.h>
#include <stdio.h>
#define print(type, x) type##_print(x)

void int64_t_print(int64_t i) { printf("%" PRId64, i); }
void double_print(double d) { printf("%lf", d); }
void print_newline() { puts(""); }

int main() {
    int64_t x = 42;
    double d = 99.99;
    print(int64_t, x);
    print_newline();
    print(double, d);
    return 0;
}
```

Question Test this program with the `-E` switch for `gcc/clang`.

Question 3 Create a macro that will output number `28134` when calling `print(2, 81, 34)`, send the code via the form. Macro should accept three parameters: a, b, c.

Generic Usage

Starting with C99, a powerful `_Generic` construct has appeared, which can branch by data types and, depending on the type of expression, substitute one or another line. If no suitable type is found in the list of types, then the line marked with `default` is substituted.

```
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
void error(const char *s) {
    fprintf(stderr, "%s", s);
    abort();
}

#define _print …//continue this line to corresponded lines in main work
properly

// Pay attention to the backslashes at the end of every line except the last
one!
// They escaped newlines, allowing you to write a macro in many lines.

#define print(x) \
```

```
    _Generic((x), \
    int64_t : int64_t_print,        \
    double : double_print, \
    default : error("Unsupported operation"))(x)

void int64_t_print(int64_t i) { printf("%" PRId64, i); }
void double_print(double d) { printf("%lf", d); }
void print_newline() { puts(""); }

int main() {

int64_t x = 42;
double d = 99.99;

print(x);

print_newline();

_print(int64_t, d);

print_newline();

print(d);

print_newline();

_print(double, d);

return 0;

}
```

Question 4 Test this program with the `-E` switch for `gcc/clang`. `Continue the grey line in order to corresponded lines in main() work properly`. Enter the macro definition to the form.

Generalized data structures (30 minutes)

There are no generic types in C. It is impossible to directly describe the data structure, say, a linked list, in which you can put numbers of any type (`int64_t`, `double` etc.)

This structure will look different for each data type:

- for `int64_t`:
- for `double`:

We can write a macro that will generate an appropriate definition:

Question 5 See the attached file sem_10_task.c. There are functions and macro definitions you should add the code to, so the output of a program will be like:

336 84 42

84 42

340.000000 85.000000 42.500000

85.000000 42.500000

Send the file with code through the form.