ECS 36A, June 5, 2023

Announcements

1. Final study guide, sample final exam, and recursion questions are posted
2. Answers to sample final are on Canvas but *not* on the nob web site
3. Extra office hours: Tu 11:00am-11:50am, Th 1:10pm–2:00pm
4. Wednesday office hour shifted to 4:10pm–5:00pm

Oops . . .

Remember *qsort*? Here is its call:

```
qsort(base, nelts, sizeof(double),
      (int (*)(const void *, const void *)) cmp);
```

I used this for *cmp*:

```
int cmp(const void *x, const void *y){
      double *px, *py;
      px = (double *)x;
      py = (double *)y;
      return(*px - *py);
}
```

What is wrong with this?

Oops . . .

It's the `*px - *py` — if it returns something less than 1.0, the function returns 0 (equal), even if there is a difference of (say) 0.5 or –0.5

```
int cmp(const void *x, const void *y){
        double *px, *py;
        px = (double *)x;
        py = (double *)y;
        if (*px > *py) return(1);
        else if (*px < *py) return(-1);
        return(0);
}
```

The lines in red replace the return in the earlier version

Last C Operator

- Abbreviated “if”

```
x = a ? b : c
```

- If `a` evaluates to non-zero, `b` is evaluated and assigned to `x`
  - `c` is ignored
- If `a` evaluates to zero, `c` is evaluated and assigned to `x`
  - `b` is ignored

Examples

```
a = 0;
b = 1;
c = 2;
x = a ? b++ : c--;
```

As `a` = 0, `c--` is evaluated, so `x` = 2 and `c` = 1

```
a = 3;
b = 1;
c = 2;
x = a ? b++ : c--;
```

As `a` ≠ 0, `b++` is evaluated, so `x` = 1 and `b` = 2

C Preprocessor

- A program that is run as part of the C compiler, *before* anything is actually compiled
- It does textual substitution only
  - It doesn't know C (or any other language for that matter)

C Preprocessor

- All lines begin with `#`
- `#define`
- `#undef`
- `#include`
- `#if, #ifdef, #ifndef`
- `#elif`
- `#else`

Example

- Suppose you will use the value of π repeatedly. Define `PI` :

```
#define PI 3.14159265
```

- Now this line

```
diameter = radius * PI;
```

- becomes this line

```
diameter = radius * 3.14159265;
```

Example

- Now suppose you will use 0 in two ways: as an end of string and as a NULL pointer

```
#define EOS  0
#define NULL ((void *) 0)
```

- Now these lines

```
*x = EOS; p = NULL;
```

- becomes these lines

```
*x = 0; p = ((void *) 0);
```

#define

```
#define BOARD  8*8
```

- Replace every occurrence of the word "`BOARD`" with "`8*8`"

- Usually used to parameterize something; examples from stdio.h:
  - NULL is a macro (0)
  - EOF is a macro (–1)
- *Warning: this is textual substitution, so do not treat them as variables!*

Watch Out For This

- Goal: create a chessboard, each side being 8 squares, and 2 extra squares for computation, for a total of 100 squares

```
#define SIDE 8+2
```

- Now every occurrence of `SIDE` is replaced by `8+2`

```
char chess[SIDE*SIDE];
```

becomes

```
char chess[8+2*8+2];
```

So the board has 26 squares

Do This

- Goal: create a chessboard, each side being 8 squares, and 2 extra squares for computation, for a total of 100 squares

```
#define SIDE (8+2)
```

Now every occurrence of `SIDE` is replaced by `(8+2)`

```
char chess[SIDE*SIDE];
```

becomes

```
char chess[(8+2)*(8+2)];
```

So the board has 100 squares

General Rule

- In the definition part of the macro, parenthesize the macro
- Without parentheses

```
#define SIDE 8+2
```

- `SIZE * SIZE = 8 + 2 * 8 + 2 = 8 + 16 + 8 = 32`
- With parentheses

```
#define SIDE (8+2)
```

- `SIZE * SIZE = (8 + 2) * (8 + 2) = 10 * 10 = 100`

Parameterized Macro

```
#define isbetween0and9(x)    ((0<=(x))&&((x)<=9))
```

- `isbetween0and9(4)` returns 1 and `isbetween0and9(-100)` returns 0
- Beware — whatever is put for `x` is evaluated every time `x` occurs in the macro definition

```
x = 9; . . . isbetween0and9(x++)
```

becomes

```
x = 9; . . . ((0<=(x++))&&((x++)<=9))
```

or

```
x = 9; . . . ((0<=(9))&&((10)<=9))
```

which returns false (as 10 > 9)

#undef

- Delete a macro definition

```
#define XYZZY    “dizzy”
. . .
#undef XYZZY
int XYZZY = -20;
```

- Without the `#undef`, the declaration becomes:

```
int “dizzy” = -20;
```

  which gives an error

#include

- Interpolate file into current source code
- When it does this, it preserves the line numbers of the original files by using these:

```
# 9 "macros.c"
```

  Next line is treated as line 9 by the compiler and debuggers
- The preprocessor inserts these lines; *you do not*

#include <*file*>

`#include <file>`

- Look for *file* in predetermined, system locations
  - Usually /usr/include, /usr/lib/include, and others
  - The “<“ “>” are what tells the C preprocessor to do this

`#include “file”`

- Look for *file* in the current working directory first
  - The quotation marks are what tells the C preprocessor to do this

`-I dir`

- Add *dir* to the list of directories to be searched
  - Look in system directories *first*, then named directories

#if...#elif...#else...#endif

- Conditional compilation

```
#if XYZZY == 1
x = 1;
#elif XYZZY == 2
x = 2;
#else
x = 0;
#endif
```

#if...#elif...#else...#endif

- If `XYZZY` is a macro
  - defined as 1, `x` will be 1
  - defined as 2, `x` will be 2
  - defined as anything else, or undefined, `x` will be 1

```
#if XYZZY == 1
x = 1;
#elif XYZZY == 2
x = 2;
#else
x = 0;
#endif
```

#ifdef, #ifndef

```
#ifdef XYZZY
. . .  compiled if XYZZY is defined (as anything)
#endif
#ifndef ABCDE
. . .  compiled unless ABCDE is defined (as anything)
#endif
```

Some Idioms

```
#ifdef notdef
. . .
#endif
```

- This comments out all code between the `ifdef` and `endif`
  - Quick way to remove code temporarily

```
#if 0
. . .
#endif
```

- This does the same thing

For Debugging

Define a debug macro like:

```
#define DEBUG
```

Then use `ifdef`s to surround debugging code

To eliminate it, just comment out the define line

Alternate Approach

Omit this line

```
#define DEBUG
```

and use the compiler command-line option `-DDEBUG`

This defines the macro `DEBUG` (set to 1)

#error

- Used to print error messages; usually to indicate that compilation will fail for some reason related to the compiler or system
- Example:

```
#ifndef unix
#error “This will only run on a UNIX system”
#endif
```