CSE 142
Computer Programming I

Program Style

Aspects of Quality Software

- Getting the syntax right
  - This may seem hard at first, but turns out to be the easiest part of all
- Getting the logic right
  - Sometimes difficult, but absolutely essential
- Today's focus: Programming with good style
  - What does this mean, and why does it matter?

Programming Style

- A program is a document:
  - Some of it is read by a computer.
  - ALL of it is read by people.
  - Donald Knuth: "literate programming"
- "Style" is a catch-all term for people-oriented programming.
  - comments, spacing, indentation, names
  - clear, straightforward, well-organized code
  - code quality

Style in This Course

- Along the way, we suggest and sometimes require particular points of style in programs that are turned in for this course.
- It is common for employers to have style requirements that all programmers must follow.

Style in This Course

- Along the way, we suggest and sometimes require particular points of style in programs that are turned in for the on campus version of this course.
- It is common for employers to have style requirements that all programmers must follow.

/* Comments */

Comment block at front of program

```
/*****************************************************
* Program:      Mi_To_Km
* Purpose:      Miles to Km  conversion
* Author:       A. Hacker, 1/18/00  Sec. AF
(Turing)
*****************************************************/
```

Comment block per major section

```
/* Calculate volume of cylinder and ...
 * Inputs:          radius, height, ...
 * Output:          volume, ...
 * Assumes:     radius, height nonnegative */
```

Small ones throughout

```
.
.
.
/* Tell user it's negative. */
```

Required Comments (1)

1. Heading comment at the beginning of each file
   - Brief explanation of what's in the file

2. Function heading comments
   - Describe *what* the function does
   - Must explain (define) all parameters and result
   - Should never have to read function body to understand how to call it

Required Comments (2)

3. Variable declaration comments
   - Describe information contained in the variable
   - Not needed for trivial variables if their usage is obvious (loop indices,etc.)
   - Should never have to read code that uses a variable to figure out what's in it

4. Statement comments
   - Higher-level summary of what the following group of statements does (as needed)
   - Say *what*, not *how*
   - Most individual statements won't need comments

Statement Comments

Say *why*, don't paraphrase the code:

NO:
```
/* subtract one from sheep */
sheep = sheep - 1;
```

YES:
```
/* account for the sheep that
   the big bad wolf just ate.*/
sheep = sheep - 1;
```


Spaces

Use blank lines to separate major sections.
Vertically align like things:

```
x       = 5 ;
yPrime  = 7 ;
z_axis  = 4.3;
```

Leave space around operators:

No: `y=slope*x+intercept;`

Yes: `y = slope * x + intercept ;`

Use parentheses for emphasis, too

Yes: `y = (slope * x) + intercept ;`

Indentation

Like an outline, indent subordinate parts
- Functions
  - Indent function body
- if statements
  - Indent what's done on true
  - Indent what's done on false (else)
- while and for loops
  - Indent loop body

Several styles are possible
- Be clear, be consistent

Identifiers (Review)

Identifiers name variables and other things
- Letters, digits, and underscores (_)
- Can't begin with a digit
- Not a reserved word like *double*, *return*

"Case-sensitive"
- *VAR*, *Var*, *var*, *vAr* are all different

Using all CAPITAL letters is legal...
- but usually reserved for *#define* constants

What's in a Name?

Extremely valuable documentation.

Microsoft Excel has over 65,000 variables.

How long is just right?

m

mph

miles_per_hour

average_miles_per_hour_that_the_red_car_went

Avoid similar names: *mph* vs. *Mph* vs. *mqh*

Suggestions for Names

Variables and value-returning functions:

- Noun phrase describing information in variable or value returned by function

Void functions:

- Verb phrase describing action performed when function is called

More Examples

OK

rectangleWidth, rectangle_Width, rectangle_width, length_10_Rectangle

Illegal

10TimesLength, My Variable, int

Legal, but bad style

a1, l, O, xggh0sxx89s,

rectangleWidth *and* rectanglewidth *or* rectangle_width

Clarity

Do "obvious" things the obvious way

No: `x = (y = x) + 1 ;`

Yes:
```
y = x ;
x = x + 1;
```

Don't be tricky, cute, or clever without GOOD reason.

If so, comment it!

#define (Review)

Named constants:

```
#define PI          3.14159265
#define HEIGHT      50
#define  WIDTH      80
#define AREA        (HEIGHT * WIDTH)
   ...
circle_area = PI * radius * radius ;
volume = length * AREA;
```

Note: = and ; are not used for #define

() can be used in #define

Using #define is Good Style

Centralize changes

No "magic numbers" (unexplained constants)

- use good names instead

Avoid typing errors

Avoid accidental assignments to constants

```
double pi ;
...
pi = 3.14 ;
...
pi = 17.2 ;
```

```
#define PI 3.14
...
PI = 17.2 ;     syntax error
```

Putting It All Together

```
/* Convert miles per hour to feet per second
 * Author:    ...
 * Date:   ...
 */

#include <stdio.h>

/* conversion constants. */
#define FEET_PER_MILE            5280.0
#define SECONDS_PER_HOUR   (60.0 * 60.0)

int  main(void)
{
    double miles_per_hour        /* input mph */
                    /* corresponding feet/sec */
    double feet_per_second;
                    /* corresponding feet/hr */
    double feet_per_hour;
```

```
    /* prompt user for input */
    printf("Enter a number of miles per hour: ");
    scanf("%lf", &miles_per_hour);

    /* convert from miles per hour to feet per
            second */
    feet_per_hour  =
          miles_per_hour  *  FEET_PER_MILE;

    feet_per_second  =
          feet_per_hour  /  SECONDS_PER_HOUR;

    /* format and print results */
    printf("%f mph is equal to %f feet per sec.\n",
            miles_per_hour, feet_per_second);

    return 0;
}
```


Many small points; Big cumulative effect...

```
#include<stdio.h>
int main(void){double v1,v2,v3,v4,v5;printf("Enter"
" a number of miles per hour:");scanf("%lf",&v1);
v5=v1*1.46666667;printf("%f miles per hour is"
" equal to %f feet per second.\n",v1,v5); return 0;}
```


Style Summary: Clarity is Job #1

DO
- Use plenty of comments - but not too many
- Use white space
- Use indentation
- Choose descriptive names
- Use named constants

DON'T
- be terse, tricky
- place speed above correctness, simplicity
- use "magic numbers"

QOTD: Guerilla Style Wars

Think of a common bug/problem you have in your code.

Now try to imagine a *stylistic* convention that would overcome that.

Example:
- I might often type = rather than ==.
- If I never write (x == 3) but instead always write (3 == x), the compiler will find my bug!

Is the 3 == x convention really good style? Would it really help? What about your convention?