Introduction To C Programming

Lesson 09

Programming Style

The purpose of programming style

- A program can use the right syntax, have no bugs, and run efficiently, but be poorly written
- Make the code easy to read and understand
  - Well written programs are easier to understand and modify
- Be consistent - use the same style throughout
  - E.g., stick with one curly brace style

What is good programming style?

- Code should be:
  - Clear and simple
  - Straightforward logic
  - Conventional language use
  - Meaningful variable, function, and constant names
  - Neat formatting
  - Helpful comments

Attributes of bad programming style

- Clever tricks
  - Clever code is often hard to understand code
  - E.g.,
    - *Using #define to make C look like another language*
    - *Complex expressions (unwrap them)*
- Unusual constructions or use of the language
  - Test: An experienced programmer has to go to a reference book to understand code
    - *for (i=0; i < COUNT; i++)*
      - **int j = 0; /* is j initialized once or for every loop? */**

Comments

```
if ( (country == SING) || (country == BRNI) ||
     (country == POL) || (country == ITALY) )
{
    /*
    ** If the country is Singapore, Brunei, or Poland
    ** then the current time is the answer time
    ** rather than the off hook time.
    ** Reset answer time and set day of week
    */
```

- What relationship links Singapore, Brunei, Poland, and Italy?
- Why isn't Italy mentioned in the comment?

Names

```
#define ONE 1
#define TEN 10
#define TWENTY 20

#define INPUT_MODE 1
#define INPUT_BUFSIZE 10
#define OUTPUT_BUFSIZE 20
```

- What are potential problems with top 3 #defines?
- Why are the next 3 better?

Names

- Variable Names
  - Use descriptive names for globals, short names for locals

```
/* Current length of input queue */
int nPending = 0;

/* Which is better? */
for (theElementIndex = 0; theElementIndex < numberOfElems;
    theElementIndex++)

/* or */

for (i = 0; i < nElemens; i++)
    elem[i] = i;
```


Names - Be Consistent

```
int UserQueue[20];
int noItemsInQ;
int queueCapacity;
```

- The word queue appears as Queue, Q, and queue
  - A reader wonders - do these all belong together?

Use active names for functions

- Function names should be based on active verbs perhaps followed by nouns
- Functions that return a boolean value should be named so that the return value is unambiguous

```
/* function with an active name */
int now = getTime();

/* ambiguous name for a boolean function */
if (checkOctal(c)) ...

/* better */
if (isOctal(c)) ...
```


Use accurate names

```
int inTable(int table[], int size, int value) {
   int j;
   int i;
   for (i = 0; i < size; i++) {
      if (table[i] == value)
         break;
   }

   return (i == size);
}
```

- Using the name it appears that inTable() returns true if the value is in the table
- In practice it does the opposite
  - This name is sure to confuse

Bad Examples

```
/* Comment on the names and values */
#define TRUE 0
#define FALSE 1

if ((ch = getchar()) == EOF)
    not_eof = FALSE;


/* Read this code aloud */
if ((falloc(SMRHSHSCRTCH, S_IFEXT | 0644, MAXRODDSHS) < 0)


/* Improve this function */
int smaller(char* s, char* t) {
    if (strcmp(s, t) < 1)
        return 1;
    else
        return 0;
}
```


Consistency

- Use consistent indentation and brace style
  - If you work on a program that is not your own - preserve the style found there

```
if (month == FEB) {
   if (year%4 == 0)
      {
      if (day > 29)
         legal = FALSE;
      }
   else {
      if (day > 28)
      {
         legal = FALSE;
      } }
}
```

Idioms

- Programming languages have conventional ways to write common code
  - i, j, k for array counters
  - p, q for short term pointer names
  - c, ch for short term char names

```
i = 0;
while (i <= n-1)
   array[i++] = 1.0;

for (i = 0; i < n; )
   array[i++] = 1.0;

for (i = n; --i >= 0; )
   array[i] = 1.0;

/* all the above are correct, but the idiomatic form is */
for (i = 0; i < n; i++)
   array[i] = 1.0;
```


Magic numbers

- Give names to magic numbers
- Define numbers as constants, not macros
- Use the language to calculate the size of an object (I.e., sizeof())
- Use character constants, not integers

```
/* what's this do?
if (c >= 65 && c <= 90)

/* a little better */
if (c >= 'A' && c <= 'Z')

/* best */
if (isupper(c))
```

Comments

- Don't belabor the obvious

```
/* default */
default:
    break;

/* return SUCCESS */
return SUCCESS

zerocount++;    /* increment zero entry counter */

/* initalize "total" to "number received"
total = number_received;
```

- Comment functions and global data
  - Make comments concise
  - Beware standard function comment header

Comments

- Don't comment bad code - rewrite it.

```
/* if "result" is 0 a match was found so return true
** (non-zero). Otherwise, "result" is non-zero so
** return false (zero).*/
#ifdef DEBUG
printf("*** isword returns !result = %d\n", !result);
fflush(stdout);
#endif

return (!result);

/* better */
#ifdef DEBUG
printf("*** isword returns matchfound = %d\n", matchfound);
fflush(stdout);
#endif

return matchfound;
```

Comment on these comments

```
void dictInsert(char* word)
/* returns 1 if word in dictionary, otherwise returns 0 */

if (n > MAX || n % 2 > 0) /* test for even number */

/*
** Write a message
** Add to line counter for each line written
*/
void writeMessage()
{
    /* increment line counter */
    line_number = line_number + 1;
    printf("%d %s\n%d  %s\n%d  %s\n",
        line_number, HEADER,
        line_number + 1, BODY,
        line_number + 2, TRAILER);

    /* increment line counter */
    line_number = line_number + 2;
}
```


Programming Style References

- Writing Solid Code, Steve McGuire, MS Press
- The Practice of Programming, Brian Kernighan, Rob Pike, Addison Wesley
- Code Complete, Steve McConnell, MS Press