Programming Style

References:

The Practice of Programming, *Brian W Kernighan and Rob Pike, Addison-Wesley, 1999*

Code Complete, *Steve McConnell, Microsoft Press, 1993*

Writing Solid Code, *Steve Maguire, Microsoft Press, 1993*

Introduction and motivation

- Style is
  - Naming
  - Comments
  - Layout (white space)
  - Conventions
  - Braces, parentheses
  - Constants

Style

- Making code fit for people to read
  - You, as you develop code
  - Next person to deal with the code
  - You, when you come back to modify code
  - Someone who wants to adapt your code
  - You, when you re-purpose and reuse
  - The marker!

Improve this

```
if ( (country == SING) || (country ==
  BRNI) || (country == POL) || (country ==
  ITALY) )
{
    /* If the country is Singapore,
     * Brunei or Poland the current
     * time is answer time, not off
     * hook time...*/
```

Kernighan&Pike, 1999

```
 if ((country == SING) ||
     (country == BRNI) ||
     (country == POL)  ||
     (country == ITALY))
{
     /* If the country is
      * Singapore, Brunei, Poland or Italy
      * the current time is answer time,
      * not off hook time...
     */
```

Naming principles

- Length matches scope
  - global descriptive, longer, comment
  - Local shorter
- Clarity of names for purpose
- Follow conventions consistently
  - Normal English spellings
  - p for pointer eg count, countp
  - UPPER CASE for constants
  - capital for globals
  - Hungarian notation g_
  - [avoid globals]

Can you improve this?

```
for ( ArrayIndex = 0; ArrayIndex <
  SizeOfArray; ArrayIndex++ )
    OrderOfElements[ArrayIndex] =
    OrderOfEElements[ArrayIndex];
```

```
for ( ArrayIndex = 0; ArrayIndex <
  SizeOfArray; ArrayIndex++ )
     OrderOfElements[ArrayIndex] =
     ExtractedElements[ArrayIndex];
```

```
for ( i = 0; i < arrSize; i++ )
     Order_elts[i] = E_elts[i];
```

```
for ( ArrayIndex = 0; ArrayIndex <
  SizeOfArray; ArrayIndex++ )
    OrderOfElements[ArrayIndex] =
    OrderOfElements[ArrayIndex];

for ( i = 0; i < arrSize; i++ )
{
    Order_elts[i] = E_elts[i];
}
```

Spot the differerence

```
VeryLongName1ForClarity
    VeryLongNam1ForClarity
 VeryLongName1ForClarity
        VeryLongNamelForClarity
InterveningSeperatorNameToo
VeryLongName1ForClarity
```

Choosing names

- Active names for functions
  - `getchar`
  - `getTime`
- Boolean functions
  - `isupper`
  - `checkId`
- Boolean names - indicate sense
  - `done`
  - `found`
  - `success`
  - `instring`

- Avoid negatives
  - `NotDone`
  - `NotFound`
- Be aware of easily-confused characters:
  - `1, l, I`
  - `0, O`
  - `2, Z`
  - `S, 5`
  - `G, 6`

Layout – white space

Whitespace principles

- Usewhitespacetoenhancereadability.
- within a line
- use blank lines for vertical spacing
- some blank lines between functions
- Be consistent on function declarations

Spacing out

- One declaration per line
- One statement per line
- Blank lines between parts of code
- The bigger the part, the more spacing around it
- Also use lines to mark out sections

Classic layout

```
/* comment explaining next part */
Code

/* comment explaining next part */
Code
```

Classic layout

```
/* -----------------------------*/
/* comment explaining next part */
Code
/* -------------------------------------------------- */
```

Classic layout

```
/* ============================ */
/* comment explaining next part */
```
Code
```
/* ============================ */
```

CExpressions and statements

- one statement per line
- **do** use redundant { }
- **do** use redundant parentheses
- **do** use spaces around operators
- **avoid** double negatives
- **do** try to be clear - not clever

Expressions and statements

```
for (i++;i<100;name[i++]='\0');
```

```
for (i++; i < 100; name[i++] = '\0')
        ;
```

```
for (i++; i < 100; i++)
    name[i] ='\0';
```

Avoid negatives (sic)

```
if ( !(age < QUAL_AGE) ||
    !(income > LO_INCOME))
  ...

if ( (age >= QUAL_AGE) ||
    (income <= LO_INCOME))
  ...
```

Parentheses v precedence

```
leap = y % 4 == 0 && y % 100 != 0 ||
          y % 400 == 0;
```

```
leap = ((y % 4 == 0) &&
   (y % 100 != 0)) || (y % 400 == 0);
```

```
leap = ((y%4 == 0) && (y%100 != 0))
  || (y%400 == 0);
```

Build code to be easily modified

- Use { } even around single controlled statements
- One declaration per line
- One statement per line
- Document things people might accidentally change due to misunderstanding

Magic numbers and constants

```
#define BMI_LO 20 /* low normal range */
#define BMI_OK 25 /* high normalrange */
#define BMI_HI 30 /* low obese range*/

enum
{
   BMI_LO = 20,   /* low normal range */
   BMI_OK = 25,   /* high normalrange */
   BMI_HI = 30    /* low obese range */
};

const int BMI_LO = 20
```

Clarity v cleverness

```
flag = flag ? 0 : 1;
```

```
if (flag)
  flag = 0;
else
  flag = 1;
```

Clarity v cleverness: Avoid side-effects

```
array[i++] = i;

array[i] = i;
i++;

scanf("%d %d", &a, &arr[a]);

scanf("%d", &a);
scanf("%d", &arr[a]);
```

Indentation

The brace debate

My preference

```
if ...
{
}
```

```
if ... {
}
```

```
if ...
    {
    }
```

dangling-else

```
if (a)
    if (b)
        x = 1;
else
    x = 2;
```

C Idiom

C idiom

```
i = 0;
while (i <= n-1)
    A[i++] = val;

for (i = 0; i < n; )
    A[i++] = val;

for (i = 0; i < n; i++)
  A[i] = val;
```

C idiom

```
Infinite loops with escape

for (;;)
  ...

while(1)
  ...
```

Comments

Principles for comments

- self-documenting code
- avoid over commenting
- comments on global variables
- comments on each function
  - preconditions
  - postconditions
- update comments to match code
- document unusual things
- but check they cannot be avoided
- comment on program with author(s)

Incomplete statements

- Make incompleteness clear
- Use indentation

```
GrossTax = income[user1] *
                taxrate
```

Summary

Make code fit for people to read

- You, as you develop code
- Next person to deal with the code
- You, when you come back to modify code
- Someone who wants to adapt your code
- You, when you re-purpose and reuse
- and …. the marker!