CODE STYLE

ADAM SWEENEY

CS 211

WICHITA STATE UNIVERSITY, EECS

```
.*/
  #include
                              <time.h>
                         _ ,o*/ <stdlib.h>
   #include/*
  #define c(C)/* - . */return (C); /*
  #include <stdio.h>/*. Moekan
   typedef/* */char p;p* u
    ][128] ,*v;typedef int ; R,i,N,I,A
                                             ,m,o,e
  [9], a[256],k [9], n[
                                256];FILE*f
 1) {; for(;
 :f) & (K>>8))^
                                        n[255 &
                              r ++ ] )]);c
                       (p*r, p*q ){ c(
                  (r ,q))}_ B(_ q){c( fseek
q))} D(){c( fclose(f ))} C( p *q){c( 0- puts(q ) )} /* /
213272*/"<outfile>" ) )u=0;i=I=(E(z[1],"rb")) ?B(2)?0:
                                           (((o
8)?(u = (p*)malloc(o))?B(0)?0:!fread(u,o,1,f):0:0)?0: D():0 ;if(
    bad\40input "));if(E(z[2],"rb")){for(N=-1;256> i;n[i++] =-1 )a[
   for(i=I=0; i<o&&(R =fgetc( f))>-1;i++)++a[R]?(R==N)?(++I>7)?(n
  )?0:(n [N ]=i-7):0: (N=R) | (I=1):0;A =-1;N=o+1;for(i=33;i<127;i+
  n[i] + 1&&N>a[i])? N= a [A=i] :0;B(i=I=0);if(A+1)for(N=n[A
δ&& (R =fgetc(f))> -1&& i <o ;i++)(i<N||i>N+7)?(R==A)?((*w
                         [I++]=i):0:0:0;D();}if(I<1)c(
  =u [i])?1:(*w[I]= 46))?(a
  " bad\401a" "yout "))for(i
                              =0;256>(R= i);n[i++]=R)for(A=
 =33;127 >R;R++)if(R-47&&R-9
v++)= (p)R;*v=0;}for(sprintf
     [i]+1;for(R
 :& R-()* w[i])*(
     /*' G*/ (*w+1,
                           "%0" "8x", x(R=time(i=0), m, o)^~
    0) ;i< 8;++ i)u [N+ i]=*(*w+i+1);for(*k=x(~
     0,i=0 ,*a);i>- 1; ){for (A=i;A<I;A++){u[+a [ A]
   ]=w[A ][e[A]] ;
                        k = [A+1]=x (k[A],a[A],a[A+1]
  );}if (R==k[I])
                     c( (E(z[3 ],"wb+"))?fwrite(
  /* */ u,o,1,f)?D
                     () | C(" \n OK."):0 :C(
  " \n WriteError"
                      )) for (i =+I-
 1 ;i >-1?!w[i][++
                       e[+ i]]:0;
  ) for( A=+i--;
                      A<I;e[A++]
  =0); (i <I-4
                      )?putchar
  (( ) 46)
                       | fflush
                       ( stdout
                       0;}c(C
                      fail")
                      dP' /
                     pd '
                      ZC
```

INTRODUCTION (RELEVANT XKCD)



...WOW.

THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.



IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.



IT'S LIKE SOMEONE TOOK A
TRANSCRIPT OF A COUPLE
ARGUING AT IKEA AND MADE
RANDOM EDITS UNTIL IT
COMPILED WITHOUT ERRORS.



WHAT IS STYLE?

- How the code looks
- Code is hard to read
 - That of other people
 - Your own code a few months later
 - Your own code a couple hours later
- It is important to write code that can be understood
- There are some generally universal standards
 - Usually a few valid options to choose from

WHY STYLE MATTERS

- "Any fool can write code that a computer can understand. Good programmers write code that humans can understand."
 - Martin Fowler, Refactoring: Improving the Design of Existing Code
- Other people will see our code
 - We'll see our code later
- It's extremely important that we write human-readable code

BASICS

INDENTATION

- Choose a size and stick with it
 - Acceptable sizes are 2, 4, or 8 columns
 - Recommend 4
- Exclusively use one of tabs or spaces
 - Many editors have settings to insert spaces or tabs by pressing the tab
 - DO NOT MIX AND MATCH TABS AND SPACES
- Indent only according to your "block-level"

NAMING

- Variables
 - Variables should be given a descriptive, succinct name, usually with an object word
 - Do NOT use Hungarian notation
- Functions
 - Function names should be given a descriptive, succinct name, usually with an action word
 - Some of this changes with Object-Oriented Programming, but we're not doing that

BUT DESCRIPTIVE NAMES ARE LONG

- Not always
 - tmp gets its point across just fine
- But there will be times where a variable or function name requires more than one word
- There are choices here
 - Lower camel case [lowerCamelCase] (my recommendation for variables)
 - Upper camel case [UpperCamelCase] (my OOP recommendation for types)
 - Lower snake case [my_function] (my recommendation for functions)
 - Upper snake case [My_Function]

CONSTANTS

- Sometimes we need to work with a number that doesn't change
- We can "name" the number by assigning it to a variable
- We also make the variable const, which means it cannot be altered
 - Constant
- const int NUM_BRANCHES = 10;
 - All caps is a style recommendation to make constants easier to distinguish from regular variables

BRACE PLACEMENT

- There are many ways to do proper brace placement
 - This class will only consider the two presented as acceptable

```
if (expression) {
    // do stuff
}

if (expression)
{
    // do stuff
}
```

STILL WITH THE BRACE PLACEMENT

- If you choose to place an open brace on its own line, nothing else goes on that line
- The same goes for the closing brace, with a couple exceptions
 - if/else if blocks
 - do/while loops

```
if (expression) {
    // do stuff
} else if (other expression) {
    // do stuff
} else {
    // do stuff
} while (expression is true);
}
```

TYPES OF COMMENTS

```
Single linestatement; // Comment// Commentstatement;
```

```
    Multiline

/*
* Beginning of comment
 * Comment body
 * Last line of comment
 */
```

WRITING GOOD COMMENTS

- Proper comments aids in the understanding of code
- Overly verbose/frequent comments or spartan comments hurt readability
- Comments should rarely discuss "what", but should discuss "how" or "why" instead
 - "What" is generally plain to see when looking at the code

OTHER TIPS

LOOPS

- By default, your loop counter variable is i
 - When dealing with loops, you may choose something more descriptive, but keep it short
 - Ex. Nested loops for rows and columns

```
for (int row = 0; row < ...) {
    for (int col = 0; col < ...) {
        // do stuff
    }
}</pre>
```

BIG IF/ELSE BLOCKS

- Sometimes in big if/else blocks, each decision only needs to do one thing
- Don't have to use braces if you don't need them
 - I recommend always using braces
 - If it's good enough for John Carmack, it's good enough for us
- But if you need them in one branch, you need them in every branch
 - This is a block-by-block decision

BIG IF/ELSE EXAMPLE

```
if (check1)
      // set var
else if (check2)
      // set var
else if (check3) {
      // set var
      // do more stuff
} else
      // set var
```

```
if (check1) {
      // set var
} else if (check2) {
      // set var
} else if (check3) {
      // set var
      // do more stuff
} else {
      // set var
```

LOGICAL GROUPS

- This one is harder to quantify
- Programs contain algorithms
 - An algorithm is a series of steps performed to complete a task
- Each step can require multiple lines of code
- Create some white space (one blank line, typically) between logical groups
- When we discuss functions, one blank line between function implementations is an example

CONSISTENCY

- The most important thing in code style is consistency
- Where you have options, make a choice and stick with it
 - I encourage trying different styles, but make your choices on a per assignment basis

REMINDER (RELEVANT XKCD)

YOUR CODE LOOKS LIKE SONG LYRICS WRITTEN USING ONLY THE STUFF THAT COMES AFTER THE QUESTION MARK IN A URL.



IT'S LIKE A JSON TABLE OF MODEL NUMBERS FOR FLASHLIGHTS WITH "TACTICAL" IN THEIR NAMES.



LIKE YOU READ TURING'S 1936 PAPER ON COMPUTING AND A PAGE OF JAVASCRIPT EXAMPLE CODE AND GUESSED AT EVERYTHING IN BETWEEN.



IT'S LIKE A LEET-SPEAK TRANSLATION OF A MANIFESTO BY A SURVIVALIST CULT LEADER WHO'S FOR SOME REASON OBSESSED WITH MEMORY ALLOCATION.

