Style Guide for C

There's no one, right way to stylize code. But there are definitely a lot of wrong (or, at least, bad ways). Even so, CS50 does ask that you adhere to the conventions below so that we can reliably analyze your code's style. Similarly do companies typically adopt their own, company-wide conventions for style.

Line Length

By convention the maximum length of a line of code is 80 characters long in C, with that being historically grounded in standard-sized monitors on older computer terminals, which could display 24 lines vertically and 80 characters horizontally. Though modern technology has obsoleted the need to keep lines capped at 80 characters, it is still a guideline that should be considered a "soft stop," and a line of 100 characters should really be the longest you write in C, else readers will generally need to scroll. If you need more than 100 characters, it may be time to rethink either your variable names or your overall design!

```
// These next lines of code first prompt the user to give two integer values and then multiplies those two integer values
together so they can be used later in the program
int first_collected_integer_value_from_user = get_int("Integer please: ");
int second_collected_integer_value_from_user = get_int("Another integer please: ");
int product_of_the_two_integer_values_from_user = first_collected_integer_value_from_user *
second_collected_integer_value_from_user;
```

In other languages, particularly JavaScript, it is significantly more difficult to constrain lines to a maximum length; there, your goal should instead be to break up lines (as via `\n`) in locations that maximize readability and clarity.

Comments

Comments make code more readable, not only for others (e.g., your TF) but also for you, especially when hours, days, weeks, months, or years pass between writing and reading your own code. Commenting too little is bad. Commenting too much is bad. Where's the sweet spot? Commenting every few lines of code (i.e., interesting blocks) is a decent guideline. Try to write comments that address one or both of these questions:

1. What does this block do?
2. Why did I implement this block in this way?

Within functions, use "inline comments" and keep them short (e.g., one line), else it becomes difficult to distinguish comments from code, even with syntax highlighting. Place the comment above the line(s) to which it applies. No need to write in full sentences, but do capitalize the comment's first word (unless it's the name of a function, variable, or the like), and do leave one space between the `//` and your comment's first character, as in:

```
// Convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

In other words, don't do this:

```
//Convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

Or this:

```
// convert Fahrenheit to Celsius
float c = 5.0 / 9.0 * (f - 32.0);
```

Or this:

```c
float c = 5.0 / 9.0 * (f - 32.0); // Convert Fahrenheit to Celsius
```

Atop your .c and .h files should be a comment that summarize what your program (or that particular file) does, as in:

```c
// Says hello to the world
```

Atop each of your functions (except, perhaps, `main`), meanwhile, should be a comment that summarizes what your function is doing, as in:

```c
// Returns the square of n
int square(int n)
{
    return n * n;
}
```

Conditions

Conditions should be styled as follows:

```c
if (x > 0)
{
    printf("x is positive\n");
}
else if (x < 0)
{
    printf("x is negative\n");
}
else
{
    printf("x is zero\n");
}
```

Notice how:

- the curly braces line up nicely, each on its own line, making perfectly clear what's inside the branch;
- there's a single space after each `if`;
- each call to `printf` is indented with 4 spaces;
- there are single spaces around the `>` and around the `<`; and
- there isn't any space immediately after each `(` or immediately before each `)`.

To save space, some programmers like to keep the first curly brace on the same line as the condition itself, but we don't recommend, as it's harder to read, so don't do this:

```c
if (x < 0) {
    printf("x is negative\n");
} else if (x < 0) {
    printf("x is negative\n");
}
```

And definitely don't do this:

```
if (x < 0)
    {
    printf("x is negative\n");
    }
else
    {
    printf("x is negative\n");
    }
```

Switches

Declare a `switch` as follows:

```
switch (n)
{
    case -1:
        printf("n is -1\n");
        break;

    case 1:
        printf("n is 1\n");
        break;

    default:
        printf("n is neither -1 nor 1\n");
        break;
}
```

Notice how:

- each curly brace is on its own line;
- there's a single space after `switch`;
- there isn't any space immediately after each `(` or immediately before each `)`;

- the switch's cases are indented with 4 spaces;
- the cases' bodies are indented further with 4 spaces; and
- each `case` (including `default`) ends with a `break`.

Functions

In accordance with C99, be sure to declare `main` with:

```
int main(void)
{

}
```

or, if using the CS50 Library, with:

```
#include <cs50.h>

int main(int argc, string argv[])
{

}
```

or with:

```
int main(int argc, char *argv[])
{

}
```

or even with:

```
int main(int argc, char **argv)
{

}
```

Do not declare `main` with:

```
int main()
{

}
```

or with:

```
void main()
{

}
```

or with:

```
main()
{

}
```

As for your own functions, be sure to define them similiarly, with each curly brace on its own line and with the return type on the same line as the function's name, just as we've done with `main`.

Indentation

Indent your code four spaces at a time to make clear which blocks of code are inside of others. If you use your keyboard's Tab key to do so, be sure that your text editor's configured to convert tabs (`\t`) to four spaces, else your code may not print or display properly on someone else's computer, since `\t` renders differently in different editors. (If using CS50 IDE, it's fine to use Tab for indentation, rather than hitting your keyboard's space bar repeatedly, since we've preconfigured it to convert `\t` to four spaces.)

Here's some nicely indented code:

```
// Print command-line arguments one per line
printf("\n");
for (int i = 0; i < argc; i++)
{
    for (int j = 0, n = strlen(argv[i]); j < n; j++)
    {
        printf("%c\n", argv[i][j]);
    }
    printf("\n");
}
```

Loops

for

Whenever you need temporary variables for iteration, use `i`, then `j`, then `k`, unless more specific names would make your code more readable:

```
for (int i = 0; i < LIMIT; i++)
{
    for (int j = 0; j < LIMIT; j++)
    {
        for (int k = 0; k < LIMIT; k++)
        {
            // Do something
        }
    }
}
```

If you need more than three variables for iteration, it might be time to rethink your design!

while

Declare `while` loops as follows:

```
while (condition)
{
    // Do something
}
```

Notice how:

- each curly brace is on its own line;
- there's a single space after `while`;
- there isn't any space immediately after the `(` or immediately before the `)`; and
- the loop's body (a comment in this case) is indented with 4 spaces.

do ... while

Declare `do ... while` loops as follows:

```
do
{
    // Do something
}
while (condition);
```

Notice how:

- each curly brace is on its own line;
- there's a single space after `while`;
- there isn't any space immediately after the `(` or immediately before the `)`; and
- the loop's body (a comment in this case) is indented with 4 spaces.

Pointers

When declaring a pointer, write the `*` next to the variable, as in:

```
int *p;
```

Don't write it next to the type, as in:

```
int* p;
```

Variables

Because CS50 uses C99, do not define all of your variables at the very top of your functions but, rather, when and where you actually need them. Moreover, scope your variables as tightly as possible. For instance, if `i` is only needed for the sake of a loop, declare `i` within the loop itself:

```
for (int i = 0; i < LIMIT; i++)
{
    printf("%i\n", i);
}
```

Though it's fine to use variables like `i`, `j`, and `k` for iteration, most of your variables should be more specifically named. If you're summing some values, for instance, call your variable `sum`. If your variable's name warrants two words (e.g., `is_ready`), put an underscore between them, a convention popular in C though less so in other languages.

If declaring multiple variables of the same type at once, it's fine to declare them together, as in:

```
int quarters, dimes, nickels, pennies;
```

Just don't initialize some but not others, as in:

```
int quarters, dimes = 0, nickels = 0 , pennies;
```

Also take care to declare pointers separately from non-pointers, as in:

```
int *p;
int n;
```

Don't declare pointers on the same line as non-pointers, as in:

```
int *p, n;
```

Structures

Declare a `struct` as a type as follows, with each curly brace on its own line and members indented therein, with the type's name also on its own line:

```
typedef struct
{
    string name;
    string dorm;
}
student;
```

If the `struct` contains as a member a pointer to another such `struct`, declare the `struct` as having a name identical to the type, without using underscores:

```
typedef struct node
{
    int n;
    struct node *next;
}
node;
```