

CS 201 C++ Style Guide

Variable Definitions

1. Use one variable per line.
2. In multi-word variable names, begin each word with a capital letter..
3. Comments should appear next to variables.
4. The "*" (pointer) and "&" (reference) modifiers should be kept with the variable name, not with the data type.

Good:

```
int NumberOfBoats;    // boats on duty
int TimeLeft;        // in seconds
int *ptr;             // look: * is with int
int &ptr;             // look: & is with int
```

Not So Good:

```
int num_boats, Num_Students;
int keepVariablesAslowerCase;
int* my_intptr;      // Keep * with name
```

Constant Definitions

Use one constant per line, and use initial caps only. ALL CAPS is too distracting. Don't use the C feature: #define.

Good:

```
const int NumberOfPlayers = 2;
const float Pi = 3.14159;
```

Not So Good:

```
const int NUM_PLAYERS = 2;
#define NUM_PLAYERS 2;
```

Comments

Use C++ style comments. Save /* */ for commenting out large chunks of code for testing purposes.

Good:

```
// Programmer Lucy Van Pelt
// CS 101
// Date: 1 June 1995
```

Not So Good:

```
/*
Programmer Sally Brown
CS 201
*/
```

Expressions (*/+/-)

All operators should be surrounded by spaces, with the exception of unary operators (operators that work on just one argument.) This includes ++, --, !, etc.

Good:

```
sum = a + b;
sum += 12;
```

Not So Good:

```
sum=a+b;
```

IF

1. Note that the closing brace is at the same level of indentation as the if it goes with.

```
if (q) {
    // some code
    // more code
}
```

2. Do not use a brace if there is only one line of code in the body of the if.

```
if (q)
    // one line of code
```

IF / ELSE

Else appears on the line after the brace that closes the if.

```
if (a == b) {
    // do this if a equals b
    //
}
else {
    // do this if a != b
    //
}
```

IF/ ELSE IF/ ELSE

Rather than nest if statements, it is often possible to write if/else if statements that do not indent deeply.

```
if (a == b) {
    // do this if a equals b
    //
}
else if (b == c){
    // do this if they don't
    //
}
else if (c == d) {
    // and so on...
    //
}
else {
    // catch all other cases that the
    // other if statements missed
}
```

FOR

Formatted like an if statement.

```
for (int i = 0; i < 10; ++i) {
    // do this
    // and do this
}
```

WHILE

Formatted like an if statement.

```
while (a == b) {
    // do this
    // and this
}
```

DO

Formatted like an if statement.

```
do {
    // all this stuff
    // and all this stuff
} while (a == b);
```

SWITCH

1. Use a `break` for each of the cases in the switch.
2. Note also the use of the default case.

```
switch (color) {
    case Green:
        // do green
        break;
    case Blue:
        // do blue
        break;
    default:
        // do default
}
```

Functions

1. Functions that return no explicit value should return `void`. Do not use `void` for functions that take no parameters:

Good:

```
void PrintReport(ReportType rep);
int SecondLeft();
```

Not So Good:

```
SendMessage(MessageType t);
bool IsTime(void);
```

2. The body of a function should be indented one tab stop with respect to the function header.

```
int PrintItems(ItemType item) {
    // code
    // code
    // code
}
```

Identifier Names

Choosing good identifier names is more of an art than a science, but there are some guidelines that may help [1].

1. **Abbreviate with care.** Don't think you're saving the reader by removing all the vowels. Some abbreviations (`min`, `max`, `num`, `ptr` -- for "pointer") are so common that they're considered OK. Long names are fine if they help readability. Context often helps determine the meaning of a name. Thus, one rule of thumb is that because global variables have little context, that their names should be more descriptive. Local variables are used in the context of the function, and therefore they can often be shorter without

loss of clarity. If you're in doubt, ask a TA. Good rule of thumb: if you can't pronounce it out loud, choose another name.

Not So Good:

```
int num_stu;
int prt_nbr;
```

Better:

```
int NumberOfStudents; // a bit much...
int NbrStudents;      // OK abbreviation
int PartNumber;
```

2. **Avoid one letter variable names** EXCEPT: There are some strong conventions for the names of things like loop counters (variables in `for` and `while` loops). C++ programmers almost always call these variables `i`, `j`, and `k` if you're using integers (which is quite common). Think of it like mathematical notation like a_i . When using temporary pointers (for stepping through strings and such) we often use `s` and `p`, unless there is an overwhelming reason not to. Of course, there are other times when one letter variable names make sense, (such as solving for coefficients of a quadratic equation), but such cases are rare.

3. **Boolean variables or function names should state a fact that can be true or false.** That way, they will read smoothly in an `if()` or `while()` statement. Use of the verb "is" can be quite helpful. For example:

Not So Good:

```
if (MissingString) {
    ...
}
```

Better:

```
if (StringIsMissing) {
    ...
}
```

4. Procedure names should reflect what they do; function names should reflect what they return.
5. Class names should be capitalized nouns. Member functions names should be verbs. For example:

```
class List {
    ...
}

List MyList;
MyList.Add(foo);
```

Wrapping Lines

We try to avoid this when at all possible, but sometimes there's nothing you can do about it. When it happens, you have two options:

1. Write helper functions or use temporary variables to hold intermediate results.
2. Break the expression up over several lines. Do this at the operator boundary, like so:

```
tax = (income * tax_rate)
      + state_tax + local_tax
      - (NumberOfDependents * Deduction);
```

Note that the operators are on the left and that the continued expression is indented.

Note that `cin` and `cout` statements that get too long can also be broken up. Note where the (single) semicolon falls.

```
cout << "At " << time
     << "The temperature is "
     << Temperature
     << " and the humidity is "
     << Humidity << "\n";
```

C Stuff Not To Use (Summary)

```
/* */ comments
printf
scanf
#define
```

References

[1] Keller, Daniel, *A Guide To Natural Naming*, SIGPLAN Notices, Vol 25, No. 5. pp. 95-102.