# Programming Project Guidelines

(Based on ***Ostermann's Programming Project Guidelines*** *(version 2.1)*)

## Originality:

Anything that you turn into my class is assumed to be solely the result of your own thought processes, research, trial and error, and creativity (Sometime we will allow a group of at most two to work on a project). The concept of *intellectual property* is particularly important in the sciences and will be emphasized in this class. If parts of the program are from someone else (including me), then those parts of the program **must** be clearly marked as being **not** your own work. You *may* not get credit for parts of a project that you were supposed to write but got from somebody else. If you turn in a project including work which is **not** your own original work and which you did **not** clearly mark as being the intellectual property of someone else, you will receive a penalty ranging from receiving a zero on the assignment to failing the class, depending on the nature of the incident.

## Project Submission:

All project submissions in my classes will be done electronically.

- To turn in an assignment, do the following (on the prime machines):
  - Set the current directory to be the directory containing the project that you want to turn in (and **ONLY** that project, please adopt a hierarchical classwork storage structure!!)
  - Run the program `442submit`, which is in the directory /home/drews/bin/ . The arguments to the program are:
    - **PROJNUM** The required argument to the program is the project number. For example, you can turn in the first project by typing `442submit 1`
    - **-f** The "`-f`" argument means *force*, meaning that the program will **delete** all previous submissions of the project before submitting the new files.
- Turn in requirements
  - You **MUST** use the program `make` to manage your projects. The `442submit` program will refuse to submit a program without a `Makefile.`
  - You must use the gcc compiler and (at least) the options `-g -Wall -O2 –Werror`
  - Programs that do not compile will not be graded and will receive zero credit (see -Werror!).

## Main program information:

The file that contains the subroutine `main` must contain a banner comment at the top listing at least:

- Your name
- The class number (442, 542)
- The name of the project

- A description of what the program does and how it works (at a high level)
- The input format
- A listing of any known bugs/problems in the program

Alternately, this information can be put into a file called `README`

## Programming Style:

To encourage the use of good programming style, all projects in this class must adhere to the following rules:

- Numeric constants are not allowed inside expressions, they must be declared as macros (or `const` types) and should be given in all upper case, as in:

```
#define MAX_NAMES 100
const int MAX_STRING 256
```

- All global variables that are **not** used outside the file must be declared as static near the top of the file, as in:

```
static int pagecount = 0;
```

- All variables that are accessed from outside the file in which they are declared must be declared in a project-wide include file, as in (from file `project.h`):

```
/* global variables */
#define MAXOFEM 256
extern int debug;
extern struct fooey AllOfEm[MAXOFEM];
```

- Each procedure call must contain at least a one line comment telling what it does and must be declared using ANSI C prototypes with explicit argument types and return value. The format that I *prefer* is:

```
/*
 *      ProcName - create a new record, initialized with arguments 'name'
 *      and 'nbytes', and return its address
 */
struct foo *
ProcName(
     char *name, /* this is what "name" means */
     int nbytes) /* this is what "nbytes" means */
```

  Unless you have a style that you prefer (and believe is better for some reason or another), I *recommend* that you follow this format closely.

- All procedure calls that are **not** used outside the file must be declared as static, as in:

```
static int
DoLocal(
     int arg) /* this is what "arg" means */
```

- All procedures local to a file must contain prototypes at the top of the file, as in:

```
/* local routines_ */
static int ParseArgs(int *pargc, char *argv[]);
static void Version(void);
```

- All procedures that are called from outside the file in which they are declared must be declared in a project-wide include file, as in (from file `project.h`):

```
/* global routines */
void Doit(void);
int DoThis(int *pargc, char *argv[]);
void DoThat(void);
```

- Programs must use standard indentation. I'm not going to specify which form to use, that's a religious issue. If you're not already comfortable with one style or another, I recommend the style in the Kernighan and Ritchie book.
- If a procedure is longer than 1 or 2 screens (or one printed page), it should probably be broken up into subroutines.
- If the lines of a procedure are indented by more than 4 or 5 "tab stops", then that probably means that the "deeply indented" part needs to be rewritten as a subroutine.
- Every program you submitted **must** have a command line parameter –debug to allow me to turn on debug information and print out meaningful debug information.

## Late Policy:

There will be a 10% per day penalty for late submissions. No assignments are accepted after 3 days delay. If you submit multiple versions for one project using –f , the last version's date is considered to use for late policy.