CS 361 – Computer Systems – Fall 2014
Homework Assignment 1
Linking - From Source Code to Executable Binary

Due: **Sunday 7 Sept. Electronic copy due at 9:00 A.M., optional paper copy may be delivered to the TA during lab.**

Overall Assignment

For this assignment, you are to write two small programs - one in C and one in C++ - and examine each of the steps in the process that converts the source code into an executable binary. Each program will consist of multiple files and each will include a variety of variables of different types and scopes, (e.g. global/local, static/auto, basic/pointers/references/user-defined), so that we can see how each of these are stored in each of the transition steps. Finally you are to answer some questions regarding your results, to demonstrate your ability to use the relevant tools.

Note: For Fall 2014, the C++ portion of this assignment is an optional enhancement.

Required Files, Components, and Variables for Part I

The first program you should write is to be written in C, and should consist of two files:

- mainC.c, containing the functions:
  - int main(int argc, char **argv);
  - double func1C(<type1> func1C_arg1, <type1> * func1C_arg2);
- func2C.c, containing
  - double func2C(<type1> func2C_arg1, <type1> * func2C_arg2);
  - static double func3C(<type1> func3C_arg1, <type1> * func3C_arg2);

Each of these files should contain the following global declarations, where <file> is replaced by the name of the file in which they are declared:

```
char <file>Hello[] = "Hello", * <file>World = "World";

<type1> <file>Global, <file>GlobalArray[N], * <file>GlobalPtr,
        <file>InitGlobal = N, <file>InitGlobalArray[N] = {N},
        * <file>InitGlobalPtr = & <file>InitGlobal;

static <type2> <file>StaticGlobal, <file>StaticGlobalArray[N],
               * <file>StaticGlobalPtr,
               <file>StaticInitGlobal = N,
               <file>StaticInitGlobalArray[N] = {N},
               * <file>StaticInitGlobalPtr = &
               <file>StaticInitGlobal;
```

Each file should also contain extern references to the (non-static) global variables declared in the other file. (Or files for part II.)

In addition, each function should contain the following local declarations, where <func> is replaced by the name of the function in which they are declared:

```
<type3> <func>Local, <func>LocalArray[N], * <func>LocalPtr,
        < func>InitLocal = N, <func>InitLocalArray[N] = {N},
        * <func>InitLocalPtr = & <func>InitLocal;

static <type4> <func>StaticLocal, <func>StaticLocalArray[N],
               * <func>StaticLocalPtr,
               <func>StaticInitLocal = N,
               <func>StaticInitLocalArray[N] = {N},
               * <func>StaticInitLocalPtr = &
               <func>StaticInitLocal;
```

The task of each function (including main) is to add up the total of all of variables to which it has access, (except hello and world)

- All functions (including main) should print the values of hello and world followed by "from <func>" as a first step.
- Main prints the total that it calculates, and all other functions return the totals. Main includes the results of calling func1 and func2 in its total, and func2 adds in the result of calling func3. Pass <file>Global and the address of <file>Global when calling the functions.
- Any variables that are not initialized should be assigned a value of N before their first use. In the case of arrays, fill the uninitialized arrays with all Ns, but initialize only the first value of the arrays to be initialized, as shown above.
- In the case of pointer variables, add up the thing pointed to by the pointers, not the values of the pointers themselves. For uninitialized pointers, make them point to any variable of the correct type

Data Types and Numerical Values

N should be 10 plus the last non-zero digit of your UIN number.

<type1> to <type4> are based on the first 4 characters of your ACCC ID (e.g. jbell) as follows:

Letter	Data type
A to F, or blank	char
G to M	int
N to S	float
T to Z, digits	double

Compilation

Once the files are written, compile each of them step by step to produce files <file>.i, <file>.s, and <file.o> containing pre-processed, compiled, and assembled versions of <file>.c respectively, and then a final executable program named mainC. Use the -E, -S, and -c flags to gcc, and also include -g to retain the maximum amount of information in the files.

Required Files, Components, and Variables for Part II (Optional Enhancement for Fall 2014)

The second program you should write is to be written in C++, and should consist of four files:

- mainCPP.c, containing the functions:
  - int main(int argc, char **argv);
  - double func1CPP(<type1> func1CPP_arg1, <type1> * func1CPP_arg2, <type1> & func1CPP_arg3);
- func2CPP.c, containing
  - double func2CPP(<type1> func2CPP_arg1, <type1> * func2CPP_arg2, <type1> & func2CPP_arg3);
  - static double func3CPP(<type1> func3CPP_arg1, <type1> * func3CPP_arg2, <type1> & func3CPP_arg3);
  - double func2CPP(<type1> * func2BCPP_arg1, <type1> & func2BCPP_arg2, <type1> func2BCPP_arg3);
- myClass.h, containing the declaration of the MyClass class
  - public class data members:

```
<type1> myclassClass, myclassClassArray[14],
    * myclassClassPtr, myclassInitClass,
    myclassInitClassArray[14], * myclassInitClassPtr;

static <type2> myclassStaticClass,
    myclassStaticClassArray[14],
    * myclassStaticClassPtr, myclassStaticInitClass,
    myclassStaticInitClassArray[14],
    * myclassStaticInitClassPtr;
```

  - public class methods:

```
MyClass( );
double method( <type1> methodArg,
  <type1> * methodArgPtr, <type1> & methodArgRef );
static double staticMethod( <type1> staticMethodArg,
  <type1> * staticMethodArgPtr,
  <type1> & staticMethodArgRef);
```

- myClass.cpp containing the method code for MyClass and the data definition for the static class variable(s).

All of the .cpp files, including myClass.cpp should have global variables declared similar to those listed above for part I. In addition, the mainCPP and func2CPP files should have global and local class variables, static and not, using only the default (no-argument) constructor. Name these variables {main|func2}{Global|Local}[Static]Obj, e.g. mainGlobalObj, func2LocalStaticObj, etc. (There are no Init versions for the class objects.)

The local variables within the class methods should follow the form for the local variables in the other functions, e.g. methodStaticInitLocal

Compile the cpp files using g++, and the -g, -E, -S, -c, and -o flags, to create an executable named mainCPP

Analysis

Once you have the programs compiled and running, you will need to examine them using readelf and objdump. Specific questions to be answered will be provided.

Required Output

- All programs should print your name and ACCC user name as a minimum when they first start.
- Beyond that, main should print out the grand total of all the summations and all functions (and methods) should print out the hello world strings as specified above. Note that the grand total should be a multiple of N.

Other Details:

- The TA must be able to build your programs by typing "make". If that does not work using the built-in capabilities of make, then you need to submit a properly configured makefile along with your source code. As always, you are free to develop wherever you wish, but the TA will be grading the programs based on their performance on the CS department machines.

What to Hand In:

1. Your code, **including a makefile if necessary, and a readme file,** should be handed in electronically using Blackboard. For this assignment, include also the completed questionnaire (see separate file) and text files with .txt extensions containing the output you used to answer the questions.
2. The purpose of the readme file is to make it as easy as possible for the grader to understand your program. If the readme file is too terse, then (s)he can't understand your code; If it is overly verbose, then it is extra work to read the readme file. It is up to you to provide the most effective level of documentation.
3. The readme file must specifically provide direction on how to run the program, i.e. what command line arguments are required and whether or not input needs to be redirected. It must also specifically document any optional information or command line arguments your program takes, as well as any differences between the printed and electronic version of your program.
4. If there are problems that you know your program cannot handle, it is best to document them in the readme file, rather than have the TA wonder what is wrong with your program.
5. A printed copy of your program, along with any supporting documents you wish to provide, (such as hand-drawn sketches or diagrams) may be handed in **to the TA** on or shortly after the date specified above. Any hard copy must match the electronic submission exactly.
6. Make sure that your **name and your ACCC account name** appear at the beginning of each of your files. Your program should also print this information when it runs.

Optional Enhancements:

It is course policy that students may go above and beyond what is called for in the base assignment if they wish. These optional enhancements will not raise any student's score above 100 for any given assignment, but they may make up for points lost due to other reasons. Note that all optional enhancements need to be clearly documented in your readme files. The following are some ideas, or you may come up with some of your own:

- For Fall 2014 the C++ portion of this assignment is an optional enhancement.
- Explore other data storage and function parameter types, such as const.
- Explore inheritance, including both overloading and overriding of methods.
- Explore public versus private versus protected.