TUTORIAL 1

DATE: 12th January 2015

---

1. Consider the programs shown in the class, with some modifications:

```
#include <stdio.h> /* included system
                      header files */
/*main.c*/
void swap (); /* declaration */
int buf [2] = {34,56}; /* initialised global */
int main ()  /* definition main */
{
  swap ();
  printf("buf[0]= %d buf[1]= %d\n", buf[0], buf[1]);
  return 0;
}
------------------------------------------------
/*swap.c*/
extern int buf []; /*declaration buf*/
#define one 1
int *bufp0 = &buf[0]; /* initialized global */
int *bufp1;   /* uninitialized global */

void swap ()   /* definition swap */
{
  int temp;    /* local */
  f();
  bufp1 = &buf[one];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
-------------------------------------------------
/*other.c*/
int buf[2];
void f()
{
   buf[0] = 3;
   buf[1] = 4;
}
```

———————————————————-

First, run `readelf` with the `-s` switch to see the symboltable of `main.o`. The fields of the output mean the following:

Num = The symbol number
Value = The address of the Symbol
Size = The size of the symbol
Type = symbol type:
Func = Function, Object, File (source file name),
Section = memory section,
Notype = untyped absolute symbol or undefined
Bind = GLOBAL binding means the symbol is visible outside the file. LOCAL binding is visible

only in the file. WEAK is like global, the symbol can be overridden.
Vis = Symbols can be default, protected, hidden or internal.
Ndx = The section number the symbol is in. ABS means absolute: not adjusted to any section address's relocation
Name = symbol name

Do a similar exercise for `swap.o` and `other.o`

Next find the relocation information for `main.o` by running `readelf` with the `-r` switch. Match it against the code obtained by doing `objdump` with the `-d` switch.

Now use the `readelf` and the `objdump` switches once again to find out how the relocatable symbols have been relocated in the executable `a.out`. Produce the excutable twice – once with and once without the switch `-static`.

2. This question assumes static linking. In each of the pairs of modules shown below, indicate how the multiply defined symbol `main` would be resolved. Your answer should be of the form "The use of the symbol `main` in module X will resolve to the declaration of `main` in module Y". You may also mention if the linker will give an error or will arbitrarily choose a declaration. 6 marks

   Module 1:

   ```
   #include <stdio.h>
   int main()
   {
     printf("%p\n", &main);
     p();
   }
   ```

   Module 2:

   ```
   #include <stdio.h>
   int main;
   int p ()
   {
     printf("%p\n", &main);
   }
   ```

   Module 1:

   ```
   #include <stdio.h>
   int main()
   {
     printf("%p\n", &main);
     p();
   }
   ```

   Module 2:

   ```
   #include <stdio.h>
   int main=1;
   int p ()
   {
     printf("%p\n", &main);
   }
   ```

   Module 1:

   ```
   #include <stdio.h>
   int main()
   {
     printf("%p\n", &main);
     p();
   }
   ```

   Module 2:

   ```
   #include <stdio.h>
   static int main=1;
   int p ()
   {
     printf("%p\n", &main);
   }
   ```

3. Kernighan's C book says – The characters /* introduce a comment, which terminates with the characters */. Comments do not nest.

   Is the description unambiguous? In any case extend the lex script example1 to introduce comments.

4. Write a lex script to find tokens in the 386-assembly produced by gcc. You can limit yourself to the tokens in the following program:

   ```
   main:
   pushl %ebp
   movl %esp, %ebp
   andl $-16, %esp
   ```

```
subl $16, %esp
call swap
movl buf+4, %edx
movl buf, %eax
movl %edx, 8(%esp)
movl %eax, 4(%esp)
movl $.LC0, (%esp)
call printf
movl $0, %eax
leave
ret
```

5. Consider a language with the following tokens:

   begin - representing the lexeme `begin`
   integer - Examples: `0, -5, 250`
   identifier - Examples: `a, A1, max`

   Construct the DFA for these tokens using the direct construction method.