# TUTORIAL 1

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

```c
#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 */
{
  static int temp = 100;     /* static variable */
  f();
  bufp1 = &buf[one];
  temp = *bufp0;
  *bufp0 = *bufp1;
  *bufp1 = temp;
}
--------------------------------------------------
/*other.c*/
int buf[2];
void f()
{
   buf[0] = 3;
   buf[1] = 4;
}
```

Now use the script called generate supplied along with this tutorial to generate all the elf files.

2. Study the following programs and their corresponding code generated. For each do the following:

   (a) Annotate fragments of target code with the source statemets that they correspond to.

   (b) Annotate each local variable and parameter with its (relative address)

   Compile your programs as:

   `gcc -static -fno-asynchronous-unwind-tables programname.c`

   (a) A program with a while loop in a file test1.c:

   ```
   int main()
   {
     int a=1,b=1;
     while(a<=10)
       {
         b=b*a;
         a++;
       }
     return b;
   }
   ```

   ```
   main:
           pushq   %rbp
           movq    %rsp, %rbp
           movl    $1, -8(%rbp)
           movl    $1, -4(%rbp)
           jmp     .L2
   .L3:
           movl    -4(%rbp), %eax
           imull   -8(%rbp), %eax
           movl    %eax, -4(%rbp)
           addl    $1, -8(%rbp)
   .L2:
           cmpl    $10, -8(%rbp)
           jle     .L3
           movl    -4(%rbp), %eax
           popq    %rbp
           ret
   ```

   (b) C Program using structures in a file test2.c

   ```
   struct data{
           int sum;
           int b[5];
           };
   ```

2

```
int main()
{
   struct data rec1;
   rec1.sum=0;
   rec1.b[0]=2;
   rec1.sum=rec1.sum+rec1.b[0];
   return rec1.sum;
}

main:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    $0, -32(%rbp)
        movl    $2, -28(%rbp)
        movl    -32(%rbp), %edx
        movl    -28(%rbp), %eax
        addl    %edx, %eax
        movl    %eax, -32(%rbp)
        movl    -32(%rbp), %eax
        popq    %rbp
        ret
```

3. 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. Verify your answer by using objdump and readelf on the .o and the a.out files.

*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);
}
```

```
#include <stdio.h>                      #include <stdio.h>
int main()                             static int main=1;
{                                      int p ()
   printf("%p\n", &main);              {
   p();                                   printf("%p\n", &main);
}                                       }
```
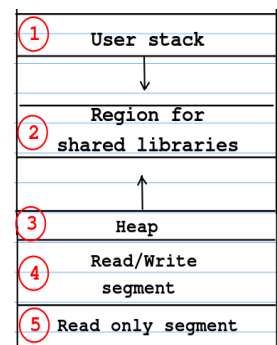
4. Consider the following modules and the memory image of a program in Linux. The regions in the memory image have been numbered. Also given are two modules in which symbols (variables and function names) have been declared. Consider the executable formed after the two modules are compiled and statically linked under the linux environment that was discussed in the class. For each symbol mention the region number in which it will be stored.

*Module 1:*

*Module 2:*

```
int x = 10;
static int y = 5;          int a[10];
main ()                    int f ()
{                          {
   int* p;                    static int k = 0;
   p = (int *) malloc(4);     return k++;
   *p = 5;                 }
   printf("%d\n", *p);
   printf("%d\n", f());
}
```

```
(1)   User stack

(2)   Region for
      shared libraries

(3)       Heap
(4)   Read/Write
      segment
(5) Read only segment
```

5. Assume that two files `test1.c` and `test2.c` contain the programs shown below:

```
test1.c:                               test2.c:

#include<stdio.h>                      #include<stdio.h>
struct {
   int x;                              struct {
   int y;                                 int y;
} rec = {13,5};                           int x;
                                       } rec;
main ()
{                                      f1 ()
   f1 ();                              {
   printf(" %d %d\n", rec.x,              printf(" %d %d\n",
          rec.y);                                 rec.x, rec.y);
}                                      }
```

What is the output of the program when the files are compiled and linked with the gcc compiler? Explain your answer.

6. This question assumes static linking. Consider the two modules shown below stored in the files f1.c and f2.c.

*Module 1:* f1.c

```
#include<stdio.h>
int a[5] = {0,1,2,3,4};
extern int b [];
double c;
int *x = &(a[3]);
int *y = b;
extern int f();
main ()
{
  c = 100.0;
  printf("%d", 5==fn());
}
```

*Module 2:* f2.c

```
int c = 58; int d = 5;
extern int a[];
int b []= {1,2,3};
int z = a[5];
fn ()
{
  int e = d;
  return(e);
}
```

Now answer the following questions assuming that an address takes 4 bytes, an int takes 4 bytes and a double takes 8 bytes.

(a) There is a compilation error in the program involving a single line. Point out the line and explain the error. For the rest of the question assume that the line is not there.

(b) The program outputs the value 0. Assuming the symbol resolution policy discussed in the class explain the output.

(c) Consider the symbol references in the program. Which of these symbol references need relocation? In which segments in the program are these relocations needed?