**NOTAS CURSO ADVANCED C PROGRAMMING**

1. ES BUENA PRACTICA DIVIDOR EL PROGRAMA EN MODULOS (code.h, code.c)
2. Los archivos .h y .c deben compartir un mismo nombre.
3. En los headers (archivos .h) se declaran y definen variables y funciones.   
   En los archivos .C se definen las funciones.
4. Tanto en el archivo main como en los otros .c se debe de agregar #include "header.h"
5. Los Makefile se utilizan para compilar el código cuando se tienen grandes programas.Evitan recompilar archivos que no han sido modificados.  
   EJEMPLO MAKE FILE:  
   SRC = mod1.c mod2.c main.c  
   OBJ = mod1.o mod2.o main.o  
   PROG = myProgram$(PROG):$(OBJ)  
    gcc $(OBJ) -o $(PROG)  
   $(OBJ): $(SRC)
6. Para comunicar archivos distintos entre sí se emplean las variables **externas (extern)**, estas son variables globales entre archivos, en cambio si se desea que una variable solo sea modificable en su respectivo archivo, esta deberá declararse **estática** **(static).**
7. **Tipos de Memoria:**
   1. **Static**: se mantiene mientras el programa se este ejecutando, se usa para almacenar variables globales o estáticas.
   2. **Stack**: almacena las variables de forma temporal, por ejemplo, variables dentro de una función (variables locales). Es del tipo LIFO por lo que cada vez que una variable se libera deja su espacio a una nueva. Tiene un limite por lo que de sobrepasarlo ocurriría un stackoverflow (por ejemplo, en problemas recursivos).
   3. **Heap**: contrario al stack, es de una estructura jerárquica (no lineal), se accede a ella por medio de los apuntares. Para asignar memoria se usa malloc() y para liberar se emplea free(). Su única restricción en cuanto al tamaño de memoria que ocupa está dada por el propio tamaño de la computadora. Sus variables pueden ser accedidas desde cualquier parte del programa.
8. **Cuando usar Heap o Stack:**
   1. **Se usa el Heap cuando:**
      1. Cuando se trabaja con arreglos o estructuras grandes.
      2. Cuando se necesita que una variable este presente por un largo periodo.
      3. Cuando se necesita un cambio dinámico del tamaño en arreglos y estructuras.
9. **Storage Class:** se usan para describir las características de una variable o una función. Nos ayudan con el seguimiento de una variable durante el transcurso de la ejecución del programa.
   1. **Auto Storage Class:** similar a las variables locales, solo se encuentran definidas en una sección del código, son automáticamente creadas cada vez que la función o bloque se ejecuta. El compilador de C asume que cada variable dentro de una función es una variable automática.
   2. **Static:** se emplea cuando se quiere que una función o variable mantenga su valor a lo largo del programa. Solo pueden inicializarse con constantes, no funciones. **NO** deben declararse dentro de una estructura debido a que estas requieren una asignación de memoria continua; sin embargo si pueden tenerse estructuras estáticas.
   3. **Register:** puede almacenar instrucciones, dirección de memoria o cualquier tipo de datos. Cuando se tiene una variable local y se quiere reducir el tiempo en el que se accede se emplea la clase register. Se usa en variables y funciones que son bastante usadas a lo largo del programa. The lifetime of a register only remains when control is within the block. El operador & no funciona con estas variables ya que no tienen asignada una dirección de memoria, se encuentran en el registro. **NO** puede usarse con static. Esta clase solo puede emplearse den manera local
10. **Lesson 50:** en C hay dos métodos para guardar información junta a modo de hacer un mejor uso de la memoria:
    1. **Bit fields y bitwise operators:** se puede utilizer una variable unsigened int/long.
    2. **Structs:** estructura del mismo tamaño que un unsigned int.
11. **Null statement (not null keyword used to allocate memory):** is needed because the compiler takes the statements that follows the looping expression as the body of the loop.
12. **I/P functions:**
    1. **Stdin:** identifica la entrada estándar del programa y normalmente se asocia con la terminal o shell, todas las funciones E/S estándar que reciben entrada de datos y no toman como argumento a un puntero de archivo (FILE pointer) obtienen su entrada de stdin.
    2. **Stdout:** igualmente es asociada con la terminal.
    3. **Stderr:** da la mayoría de los mensajes de error, también asociada con la terminal.