University of Texas at El Paso
Computational Science (CPS)

C Code In Linux

```
 -----------------------------------------------------------
CPS_array_1D.c
compile : % gcc CPS_array_1D -o  out
Execute:  %./out
-----------------------------------------------------------

#include <stdio.h>
#define SIZE 3

int main(void) {
    int i;           /* counter variable */
    int array[SIZE]; /* declare an array of 3 ints */

    printf("Displaying Array:\n");
    for (i = 0; i < SIZE; i++)
        array[i] = i + 5;  /* any other numbers desired */

    printf("Show matrix array:\n");
    for (i = 0; i < SIZE; i++) {
        printf("%d\n", array[i]);
    }

    return 0;
}


-----------------------------------------------------------
CPS_array_2D.c
compile : % gcc CPS_array_2D -o  out
Execute:  %./out
-----------------------------------------------------------

/*To pass two-dimensional array to a function as an argument,
 starting address of memory area reserved is passed as in one dimensional array */

#include <stdio.h>
#define SIZE 3

void Function(int c[SIZE][SIZE]); //void Function( );

int main() {
   int c[SIZE][SIZE], i, j;

   Function(c); /* passing multi-dimensional array from function */

   printf("Creat Array:\n");

   for (i = 0; i < SIZE; ++i) {
       for (j = 0; j < SIZE; ++j) {
           printf("%d ",c[i][j]);
       }
      printf("\n");
  }
   return 0;
}

void Function(int c[SIZE][SIZE]){
/* Instead to above line, void Function(int c[][SIZE]){ is also valid */
  int i, j;

  printf("Show matrix array:\n");
  for (i = 0; i < SIZE; ++i)
     for (j = 0; j < SIZE; ++j)
         c[i][j] = 1;
 }

-----------------------------------------------------------
CPS_array_function_1D.c
compile : % gcc CPS_array_function_1D -o  out
Execute:  %./out
-----------------------------------------------------------

#include <stdio.h>
#define SIZE 3

/* function prototype; ar[] being a pointer to type int; n being array size */
void func (); //void func(int ar[], int n);

int main(void) {
    int i;           /* counter variable */
    int array[SIZE]; /* declare an array of 3 ints */

    func(array, SIZE);/* call func() to initialise array[]; pass array name, itself a pointer and an array size */

    printf("Show matrix array:\n");

    for (i = 0; i < SIZE; i++) {
        printf("%d\n", array[i]);
    }

    return 0;
}

void func (int ar[], int n) {
    int i;
    for (i = 0; i < n; i++)
        ar[i] = i + 5;  /* any other numbers desired */
}

-----------------------------------------------------------
CPS_array_function_2D.c
compile : % gcc CPS_array_function_2D -o  out
Execute:  %./out
-----------------------------------------------------------
/*To pass two-dimensional array to a function as an argument,
 starting address of memory area reserved is passed as in one dimensional array */

#include <stdio.h>
#define SIZE 3

void Function(int c[SIZE][SIZE]); //void Function( );

int main() {
   int c[SIZE][SIZE], i, j;

   Function(c); /* passing multi-dimensional array from function */

   printf("Creat Array:\n");

   for (i = 0; i < SIZE; ++i) {
       for (j = 0; j < SIZE; ++j) {
           printf("%d ",c[i][j]);
       }
      printf("\n");
  }
   return 0;
}

void Function(int c[SIZE][SIZE]){
```

```
/* Instead to above line, void Function(int c[][SIZE]){ is also valid */
  int i, j;

  printf("Show matrix array:\n");

  for (i = 0; i < SIZE; ++i)
     for (j = 0; j < SIZE; ++j)
         c[i][j] = 1;
 }
-----------------------------------------------------------
CPS_sizeof_array_function_pointer_1D.c
Compile: $ gcc CPS_sizeof_array_function_pointer_1D -o out
Execute: $./out
-----------------------------------------------------------

#include <stdio.h>

void PrintSize_1(int p_someArray[10], int m);
void PrintSize_2(int *p_someArray, int m);

int main () {
    int i, myArray[10];
    int m = sizeof(myArray)/sizeof(int);

    printf("array size = %zu\n", sizeof(myArray)/sizeof(int)); /* as expected 10 */
    printf("array size* number of bits = %zu\n", sizeof(myArray)); /* as expected 40 */

    printf("\narray elements\n"); /* as expected 40 */

    for(i = 0 ; i < m ; i++) {
myArray[i] = i*10;
printf("%d  ", myArray[i]);
     }

    PrintSize_1(myArray, m);/* prints 4 not 40 */
    PrintSize_2(myArray, m);/* prints 4 not 40 */
}

/* array-type is implicitly converted into pointer type when you pass it in to a function.*/
void PrintSize_1(int p_someArray[10], int m){
    int i;
    printf("\n%zu\n", sizeof(p_someArray));
    printf("%zu\n", sizeof(p_someArray)/sizeof(int));

    for(i = 0 ; i < m ; i++) {
printf("%d  ", p_someArray[i]);
     }
}

// so, are equivalent. So what you get is the value of sizeof(int*)
void PrintSize_2(int *p_someArray, int m){
    int i;
    printf("\n%zu\n", sizeof(p_someArray));
    printf("%zu\n", sizeof(p_someArray)/sizeof(int));

    for(i = 0 ; i < m ; i++) {
printf("%d  ", p_someArray[i]);
    }
}

-----------------------------------------------------------
CPS_sizeof_array_function_pointer_2D.c
Compile: $ gcc CPS_sizeof_array_function_pointer_2D -o out
Execute: $./out 3 3
-----------------------------------------------------------
/*number of rows and columns of a 2D Array from a double pointer pointed to the array.*/

#include <stdio.h>
```

```
#include <stdlib.h>

void get_details(int **a, int ROW, int COL) {
 printf("\n array size =  %d x %d\n", ROW, COL);
}

int main(int argc, char *argv[]) {

 int m = atoi(argv[1]);
 int n = atoi(argv[2]);

 int i, j;
 int **a = (int **) malloc(n * sizeof(int *)); // using a double pointer

 for(i = 0; i < n; i++)
   a[i] = (int *) malloc(n * sizeof(int));

 printf("Number of [columns, rows] = [ %d  %d]\n", m , n);
 //scanf("%d %d", &m, &n);

 /*for(i = 0; i < n;i++)
     for(j = 0; j < n;j++) {
         printf("\nEnter Element %d x %d: ",i , j);
         scanf("%d", &a[i][j]);
     }
    */

 for(i = 0 ; i < m ; i++) {
     for(j = 0 ; j < n ; j++) {
a[i][j] = i*10 + j;
printf("%d   ", a[i][j]);
     }
     printf("\n");
 }

 printf("column = %zu\n", sizeof(a));
 printf("column = %zu\n", sizeof(a)/sizeof(int) );

 get_details(a, m, n) ;

 return 0;
}

-----------------------------------------------------------
Compile multiple files in a program. Wrote these three diferent programs separeted (boolean.c, barith2.c and boolean.h).
boolean.h is a header file. A header file is a file with extension .h which contains C function declarations and macro
definitions and to be shared  between  several source files. You request the use of a header  file in your  program by
including it, with the C preprocessing directive "#include"

Compile: $ gcc boolean.c  barith2.c  -o out
Execute: $./out
-----------------------------------------------------------
/**************************************************************
 * File:    boolean.c
 * Purpose: Implementation of functions to report the results of boolean operations.
 * compile: gcc boolean.c barith2.c -o out
 **************************************************************/

#include <stdio.h>
#include "boolean.h"

void And (int A, int B) {
  int Result;

  Result = A && B;
  printf ("%d && %d = %d\n", A, B, Result);
}
```

```
void Or (int A, int B) {
  int Result;

  Result = A || B;
  printf ("%d || %d = %d\n", A, B, Result);
}

void EQ (int A, int B) {
  int Result;

  Result = A == B;
  printf ("%d == %d = %d\n", A, B, Result);
}

void NE (int A, int B) {
  int Result;

  Result = A != B;
  printf ("%d != %d = %d\n", A, B, Result);
}

void Not (int A) {
  int Result;

  Result = ! A;
  printf ("! %d = %d\n", A, Result);
}

/**************************************************************
 * File:    barith2.c
 * Purpose: Program to report the results of boolean operations.
 **************************************************************/

#include "boolean.h"

int main () {

  And (0, 0);
  And (0, 1);
  And (1, 0);
  And (1, 1);
  Or (0, 0);
  Or (0, 1);
  Or (1, 0);
  Or (1, 1);
  EQ (0, 0);
  EQ (0, 1);
  EQ (1, 0);
  EQ (1, 1);
  NE (0, 0);
  NE (0, 1);
  NE (1, 0);
  NE (1, 1);
  Not (0);
  Not (1);

  return 0;
}
/**************************************************************
 * File:    boolean.h
 * Purpose: Interface for functions to report the results of boolean operations.
 **************************************************************/

#ifndef boolean_h
#define boolean_h

void And (int A, int B);
void Or (int A, int B);

void EQ (int A, int B);

void NE (int A, int B);

void Not (int A);

#endif
```