CSC180: Lecture 18

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Pointers && DMA && Arrays - example 1

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 10    //Size of 1D Array

int main ()
{
int  *p;
int  i;

p = (int *) malloc ( SIZE * sizeof (int) ); //Dynamic Memory Allocation of 1D Array

if (p == NULL) //Incase of memory allocation failure execute the error handling code block
 {
    printf ("\nOut of Memmory");
    exit (1);
 }

for (i = 0; i<SIZE; i = i + 1)
 {
       p [i] = i; // Loading the Array
 }

for (i = 0; i<SIZE; i = i + 1)
 {
       printf ("\n%d", *(p + i)); // Displaying the Array
}
    free( p );
   return 0;
}
```

Pointers && DMA && Arrays - example 2

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE1 10        //Number of Rows in 2D Array
#define SIZE2 10        //Number of Columns in 2D Array

int main ()
{
int i,j;

int (*p) [SIZE1]; //Observe the special pointer declaration to tell compiler 2D array has Size1 rows


p = (int (*) [SIZE1]) malloc ( SIZE1 * SIZE2 * sizeof (int) ); //Dynamic Memory Allocation of 2D Array


if (p == NULL) //Incase of memory allocation failure execute the error handling code block
  {
    printf ("\nOut of Memory");
    exit (1);
  }

for (j = 0; j<SIZE1; j++)
  {
   for (i = 0; i<SIZE2; i++)
    {
       p [j] [i] = j*SIZE1 +i; // Loading the Array
    }
  }
```

Pointers && DMA && Arrays - example 2 – cont’d

```
for (j = 0; j<SIZE1; j++)
 {
  for (i = 0; i<SIZE2; i++)
   {
      printf ("\n%d",p [j] [i]); // Displaying the Array
   }
 }
free ( p );
return 0;
}
```

Pointers && DMA && Arrays - example 3

```
#include <stdio.h>
#include <stdlib.h> /* required for the malloc and free functions */

int main() {
  int number;
  int *ptr;
  int i;

  printf("How many ints would you like store? ");
  scanf("%d", &number);

  ptr = malloc(number*sizeof(int)); /* allocate memory */

  if(ptr!=NULL) {
    for(i=0 ; i<number ; i++) {
      *(ptr+i) = i;
    }

    for(i=number ; i>0 ; i--) {
      printf("%d\n", *(ptr+(i-1))); /* print out in reverse order */
    }

    free(ptr); /* free allocated memory */
    return 0;
  }
  else {
    printf("\nMemory allocation failed - not enough memory.\n");
    return 1;
  }
}
```

Pointer Pitfall 1: uninitialized pointer

```
#include <stdio.h>


int main ()
{
 int a, *p;
 a = 1;
 *p = a;
 return 0;
}
```

Pointer Pitfall 1: correction

```
#include <stdio.h>


int main ()
{
 int a, *p = NULL;
 a = 1;
 *p = a; //This would generate a run
         // time error always and a compile time
         // error in smart compilers.
 return 0;
}
```

Pointer Pitfall 2: incorrect pointer usage

```
#include <stdio.h>
#include <alloc.h>

int main ()
{
 int a, *p = NULL;
 a = 1;
 p = a;
 printf ("%d",*p);          //This will print garbage value
 return 0;
}
```

Pointer Pitfall 2: correction

```
#include <stdio.h>

int main ()
{
 int a, *p = NULL;
 a = 1;
 p = &a;
 printf ("%d",*p);
 return 0;
}
```