CSC180: Lecture 17

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

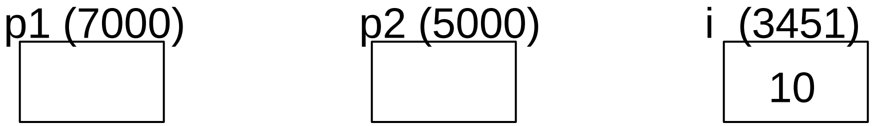
Acknowledgement: These slides are partially based on the slides supplied
with Prof. Savitch book: Problem Solving with C

Pointer multiple indirection

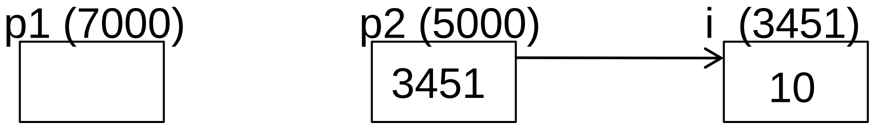
- in C it is permitted for a pointer to point to another pointer.
  - As a result many layers of pointer can be formed and this called multiple indirection
  - A pointer to a pointer has declaration similar to that of a normal pointer but have more asterix * sign before them indicating the depth of the pointer.

Pointer multiple indirection - example

```
#include <stdio.h>

int main ()
{
   int i = 10;
   int **p1
   int  *p2;

   p2 = &i;
   p1 = &p2; // Multiple indirection

   printf (" **p1 = %d And *p2 = %d", **p1,*p2);      //Statement will show 10 twice.

   return 0;
}
```



Pointer multiple indirection - example



p1 (7000)

p2 (5000)

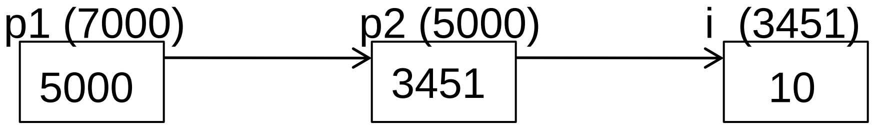
3451

i (3451)

10

```
#include <stdio.h>

int main ()
{
   int i = 10;
   int **p1
   int  *p2;

   p2 = &i;
   p1 = &p2; // Multiple indirection

   printf (" **p1 = %d And *p2 = %d", **p1,*p2);      //Statement will show 10 twice.

   return 0;
}
```

Pointer multiple indirection - example



```
#include <stdio.h>

int main ()
{
   int i = 10;
   int **p1
   int  *p2;

   p2 = &i;
   p1 = &p2; // Multiple indirection

   printf (" **p1 = %d And *p2 = %d", **p1,*p2);      //Statement will show 10 twice.

   return 0;
}
```

Pointer comparison

- Two pointers can be compared no matter where they point.
- Comparison can be done using <, >, =, <= and >= operators.
- Though it is not forcibly implied but comparison of two pointers become sensible only when they are related such as when they are pointing to element of same arrays.

Pointer comparison - example

```
#include <stdio.h>

int main ()
{
   int data[100];
   int *p1;
   int *p2;
   int i;

   for (i = 0; i <100; i = i +1 )
    {
      data[i] = i;
    }



   p1 = &data [1];
   p2 = &data [2];

    if (p1 > p2)
       {
           printf ("\n\n p1 is greater than p2");
       }
     else
       {
          printf ("\n\n p2 is greater than p1");
       }
}
```

Pointer and arrays

- Array and Pointers in c are very closely related. Infact they are so similar to each other in nature that they can be used interchangeably in each other positions most of the time.

- Important link joining them is that array name without the brackets is the pointer name and other end a pointer can be indexed as if its an array.

Pointer and arrays - example

```
#include <stdio.h>

int main ()
{
   int data[100];
   int* p1;
   int i;
   for (i = 0; i <100;i  = i  + 1) {
      data[i] = i;
    }

   p1 = data; //Assigning base address of an array to pointer

   for (i = 0; i <100;i  = i  + 1) //Accessing Array using index
   {
      printf ("\n%d",p1[i]);
    }

  for (int i = 0; i <100;i++) //Access Array using Pointer Arithmetic
   {
      printf ("\n%d",*(p1 +i));
    }

return 0;
}
```

Pointer and arrays – cont’d

- Pointers like any other data type can be arrayed. called array of pointers

- *Array of pointers are declared as shown below:*

```
data_type    *variable_name [array_size];
```

E.g.

```
int      *parrnValues[10];
char     *parrcValues[100];
```

Pointer and arrays – example 2

```
#include <stdio.h>

int main ()
{
   int data[5];
   int *array[5];
   int i;

   for (i = 0; i <5; i= i +1)
    {
      data[i] = i;
    }

   for (i = 0; i <5; i = i +1 ) //Assigning address of elements of array data to array of pointers.
    {
      array[i] = &data[i];
    }

   for (i = 0; i <5; i = i + 1) //Accessing Array value using index
    {
      printf ("\n%d",data[i]);
    }

  for (i = 0; i <5; i = i + 1) //Access Array value using array of pointers
    {
      printf ("\n%d",*array[i] );
```

Pointer & dynamic memory allocation

- Dynamic memory allocation (DMA)
  - Sometimes Memory requirement cannot be known at compile time but depends upon the input user gives interaction or some other dynamic values which keeps changing.
  - In such cases memory requirement of the program may expand or shrink at run time and in this DMA comes handy.

Pointer & dynamic memory allocation

- Dynamic memory allocation (DMA) , how to ?
  - Reserve the needed memory at run time when you need it
  - Return the memory back when you are done

Pointer & DMA functions: malloc

- void* malloc (int number_of_bytes)
  - malloc stands for *memory allocations* and is used to allocate number_of_bytes from computer memory
  - Returns a pointer to the beginning of the allocated memory

defined in the standard library header <stdlib.h>

Pointer & DMA functions: free

- void free (void *p)
  - used to return allocated memory from malloc back to heap

defined in the standard library header <stdlib.h>

Pointer & DMA functions: sizeof

- int sizeof (typename)
  - used to return the number of bytes that the underlying system reserves for a specific type
  - E.g.

```
int nBytesInInt    = sizeof( int );
int nBytesInfloat = sizeof( float );
int nBytesInInt    = sizeof( int );
```

defined in the standard library header <stdlib.h>

Pointers && DMA functions example 1

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

int main ()
{

int *p;


p = (int *) malloc ( sizeof (int) ); //Dynamic Memory Allocation


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

*p = 100;

printf ("\n p = %d", *p);   //Display 100 of course.

return 0;
}
```

Pointers && DMA functions example 1

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

int main ()
{

int *p;


p = (int *) malloc ( sizeof (int) ); //Dynamic Memory Allocation


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

*p = 100;

printf ("\n p = %d", *p);   //Display 100 of course.

free (p );

return 0;
}
```

Is there something missing here?

Pointer & DMA & Dynamic Arrays

- Normal arrays can be increased in power and flexibility using DMA to be converted into dynamic allocated arrays.

- These dynamic allocated arrays though have a little bit of complication involved with them in usage, so read carefully the explanation given below. Also their declaration varies entirely.

Pointer & DMA & Dynamic Arrays: sizeof

- int sizeof (typename)
  - used to return the number of bytes that the underlying system reserves for a specific type
  - E.g.

```
int nBytesInInt    = sizeof( int );
int nBytesInfloat = sizeof( float );
int nBytesInInt    = sizeof( int );

int nBytesInArrayof5Ints = sizeof( int ) * 5;
```

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;
}
```