Computer Programming 143 – Lecture 21
Pointers IV

Electrical and Electronic Engineering Department
University of Stellenbosch

Prof Johan du Preez
Mr Callen Fisher
Dr Willem Jordaan
Dr Hannes Pretorius
Mr Willem Smit

Copyright & Disclaimer

Copyright

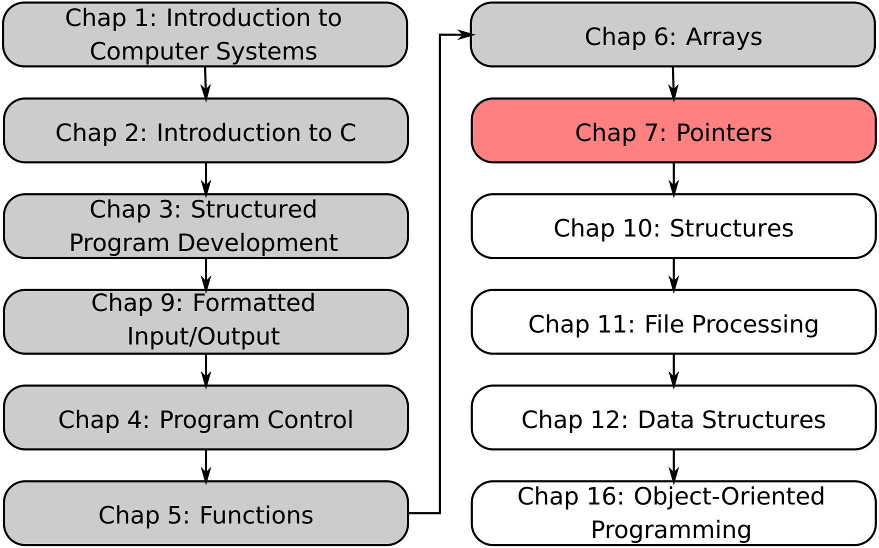
Disclaimer

This content is provided without warranty or representation of any kind. The use of the content is entirely at your own risk and Stellenbosch University (SU) will have no liability directly or indirectly as a result of this content.

The content must not be assumed to provide complete coverage of the particular study material. Content may be removed or changed without notice.

The video is of a recording with very limited post-recording editing. The video is intended for use only by SU students enrolled in the particular module.

Module Overview


Chap 1: Introduction to Computer Systems
Chap 2: Introduction to C
Chap 3: Structured Program Development
Chap 9: Formatted Input/Output
Chap 4: Program Control
Chap 5: Functions
Chap 6: Arrays
Chap 7: Pointers
Chap 10: Structures
Chap 11: File Processing
Chap 12: Data Structures
Chap 16: Object-Oriented Programming

Lecture Overview

1. Pointer Expressions and Pointer Arithmetic (7.8)
2. The Relationship Between Pointers and Arrays (7.9)
3. Dynamic Memory Allocation (12.3)

7.8 Pointer Expressions

Arithmetic operations can be performed on pointers

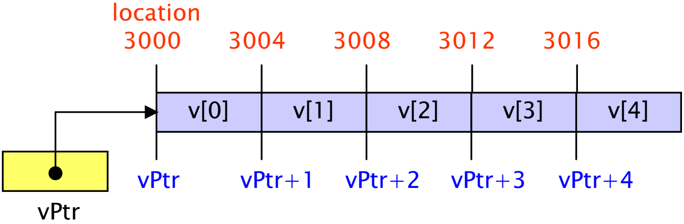
- Increment/decrement pointer (++ or - -)
- Add an integer to a pointer(+ or += , - or -=)
- Pointers may be subtracted from each other
- Operations meaningless unless performed on an array

7.8 Pointer Expressions (cont...)

5-element int array on computer with 4-byte ints

```
int v[5];
int *vPtr = &v[0]; /* or int *vPtr = v; */
```

- vPtr points to first element v[0]
  - at address 3000 (vPtr == 3000)
- vPtr ++; sets vPtr to 3004
- vPtr += 2; sets vPtr to 3008
  - vPtr points to v[2] (incremented by 2), but the machine has 4-byte ints, so it points to address 3008



7.8 Pointer Expressions (cont...)

Subtracting pointers

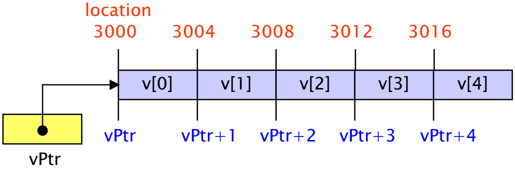
- Returns number of elements from one to the other.

```
int v[5], *vPtr0, *vPtr2; /* correct multi-pointer declaration */
vPtr0 = &v[0];
vPtr2 = &v[2];
```

`vPtr2 - vPtr0` would produce 2

Pointer comparison (<, == , >)

- See which pointer points to the earlier/later numbered array element
- `(vPtr+1) > vPtr` will return true
- Also, see if a pointer points to nothing (NULL or 0)



7.8 Pointer Expressions (cont...)

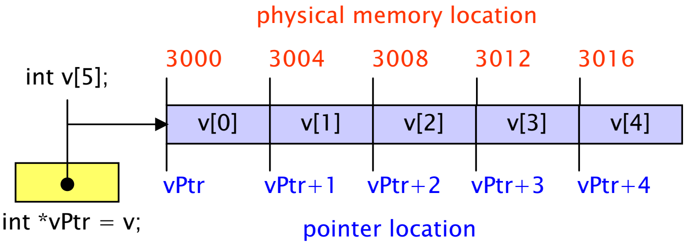
Pointers of the same type can be assigned to each other

- If not the same type, a cast operator must be used

```c
int *vPtr0;
char *vPtr2;
vPtr2 = (char*)vPtr0;
```

- Exception: pointer to void (`type void *`)
  - Generic pointer, represents any type
  - No casting needed to convert a pointer to void pointer
  - void pointers cannot be dereferenced

7.9 Pointers and Arrays



Element v[3]

- Can be accessed by ∗(vPtr + 3)
  - Where 3 is the offset. Called pointer/offset notation
- Can be accessed by vPtr[3]
  - Called pointer/subscript notation
  - vPtr[3] same as v[3]
- Can be accessed by performing pointer arithmetic on the array itself
  - ∗(v + 3)

```c
/* Using subscripting and pointer notations with arrays */

#include <stdio.h>

int main()
{
   int b[] = { 10, 20, 30, 40 }; /* initialize array b */
   int *bPtr = b;                /* set bPtr to point to array b */
   int i;                        /* counter */
   int offset;                   /* counter */

   /* output array b using array subscript notation */
   printf( "Array b printed with:\nArray subscript notation\n" );
   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "b[ %d ] = %d\n", i, b[ i ] );
   } /* end for */

   /* output array b using array name and pointer/offset notation */
   printf( "\nPointer/offset notation where\n"
           "the pointer is the array name\n" );
   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( b + %d ) = %d\n", offset, *( b + offset ) );
   } /* end for */

   /* output array b using bPtr and array subscript notation */
   printf( "\nPointer subscript notation\n" );
   /* loop through array b */
   for ( i = 0; i < 4; i++ ) {
      printf( "bPtr[ %d ] = %d\n", i, bPtr[ i ] );
   } /* end for */

   /* output array b using bPtr and pointer/offset notation */
   printf( "\nPointer/offset notation\n" );
   /* loop through array b */
   for ( offset = 0; offset < 4; offset++ ) {
      printf( "*( bPtr + %d ) = %d\n", offset, *( bPtr + offset ) );
   } /* end for */

   return 0;
}
```

```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40

Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Pointer/offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

12.3 Dynamic Memory Allocation I

malloc()

- Contained in stdlib.h
- Allocates memory during execution time

-
  ```
  newPtr = malloc( numberOfElements * sizeof( int ) );
  ```
  - Allocates memory for an array with numberOfElements number of **int** elements
  - Starting address of memory block is stored in newPtr

free()

- Contained in stdlib.h
- Frees memory allocated previously
- Always free dynamically allocated memory to prevent memory leaks

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

int main( void )
{
   int size; //number of memory units needed
   int counter;
   float *newPtr; // pointer to a float;
   float *myFloat; // pointer to a float
   float *myArray; // pointer for our "dynamic" array

   /*allocates memory using ’malloc’ and set our float
   pointer to newPtr*/
   newPtr = malloc( sizeof( float ) );
   myFloat = newPtr;
   *myFloat = 42.13;

   printf( "Enter the number of array elements: " );
   scanf( "%d", &size );
   /*allocates memory for an array with 'size' number of type float
   elements and set our array pointer to newPtr*/
   newPtr = malloc( size * sizeof( float ) );
   myArray = newPtr;

   // assigns values to array elements
   for ( counter = 0; counter < size; counter++ ) {
       myArray[ counter ] = (float)counter/2;
   }
   // displays all the array elements
   printf( "\nArray is:\n" );
   for ( counter = 0; counter < size; counter++ ) {
       printf( "%3.2f  ", myArray[ counter ] );
       if ( ( counter + 1 ) % 20 == 0 ) {
            printf( "\n" );
       }
   }
   free( myFloat ); // frees the memory allocated to 'myFloat'
   free( myArray ); // frees the memory allocated to 'myArray'
   return 0; // indicates successful termination
} // end main
```

Perspective

Today

Pointers IV

- Pointer expressions and arithmetic
- Pointers and arrays
- Dynamic memory allocation

Next lecture

Pointers V

Homework

1. Study Sections 7.8-7.9, 12.3 in Deitel & Deitel
2. Do Self Review Exercises 7.2, 7.3 in Deitel & Deitel
3. Do Exercises 7.9, 7.21 in Deitel & Deitel