C Programming Basics

SDS 322/329

November 12, 2015

Email any questions to:
rauta@tacc.utexas.edu

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Recap & Administrative Trivia

- Functions in C
  - Recursion
  - In-class exercises on recursion and implementing the strcpy() function

- Mid-term exam – grading done - will be returned today

Overview of the Course

- Writing a Basic C Program
- Understanding Errors
- Comments Keywords, Identifiers, Variables
- Operators
- Standard Input and Output (Basic)
- Control Structures
- Standard Input and Output
- Arrays, Structures
- Functions in C
- Pointers
- Working with Files

All the concepts will be accompanied with examples.

Pointers

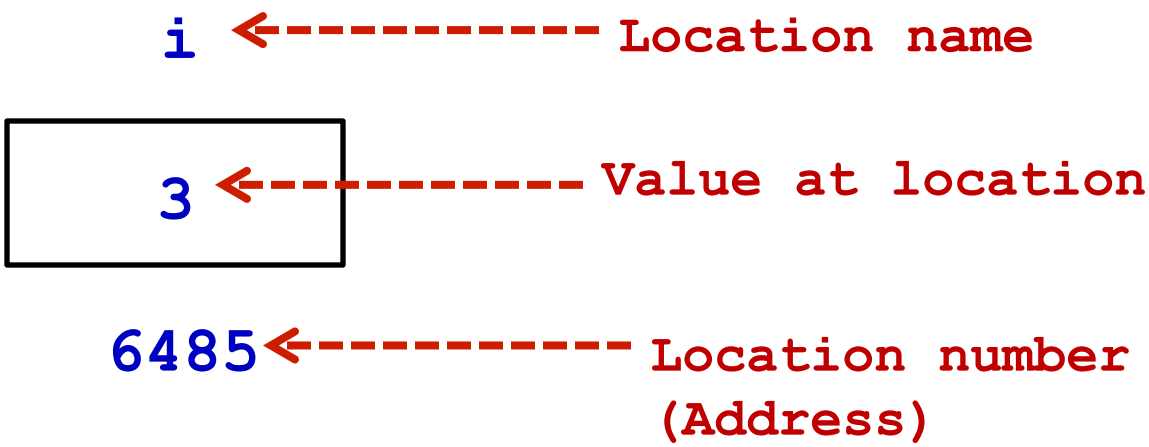
- A pointer is a variable that stores an address in memory - address of another variable
- For instance, the value of a pointer may be 42435. This number is an address in the computer's memory which is the start of some data
- We can dereference the pointer to look at or change the data
- Like variables, you have to declare pointers before you use them
- The data type specified with pointer declaration is the data type of the variable the pointer will point to

Revisiting Variable Declaration

- Consider the declaration

```
int i = 3;
```

- This declaration tells the C compiler to:
  - Reserve space in memory to hold the integer value
  - Associate the name `i` with this memory location
  - Store the value `3` at this location



‘Value at Address’ Operator: printAddress.c

```
#include <stdio.h>
int main(){
  int i=3;
  printf("\nAddress of i = %u", &i);
  printf("\nValue of i = %d", i);
  printf("\nValue of i = %d", *(&i));
  return 0;
}
```

& operator is ‘address of’ operator

* operator is ‘value at address of’ operator

Output:

```
Address of i = 2293532
Value of i = 3
Value of i = 3
```

Note:

&i returns the address of variable i

***(&i) returns the value at address of i**

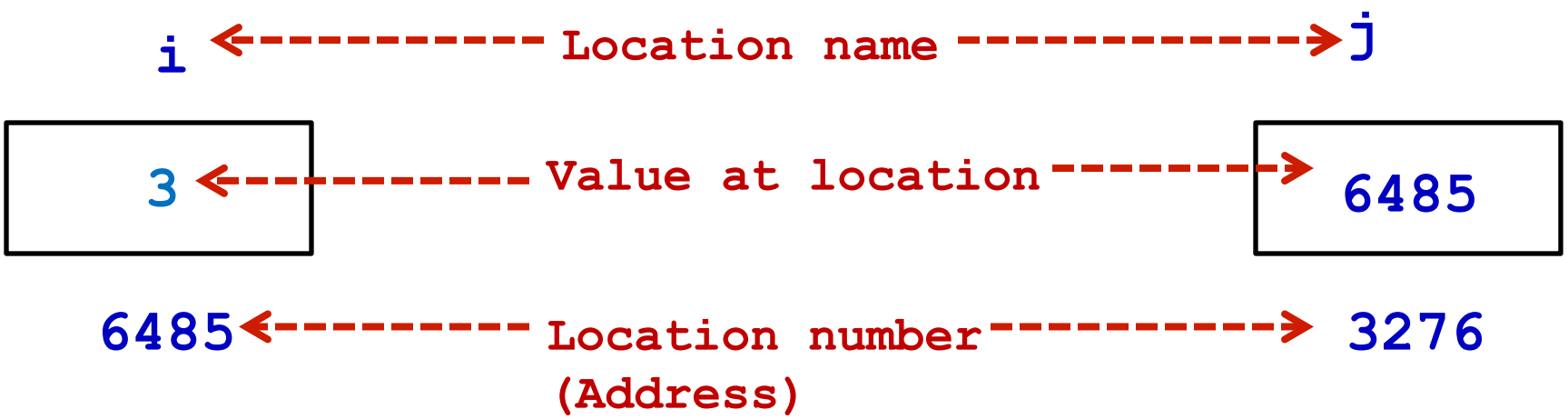
THE UNIVERSITY OF TEXAS AT AUSTIN

Pointer Expressions

- In the previous example, the expression **`&i`** returns the address of **`i`**.
- This address can be collected in a variable as

```
j = &i;
```

- **`j`** is a variable which contains the address of another variable and is declared as **`int *j;`**



Pointers: pointerExample2.c

```
#include <stdio.h>
int main(){
 int i=3;
 int *j;
 j = &i;
 printf("\nAddress of i = %u", &i);
 printf("\nAddress of i = %u", j);
 printf("\nAddress of j = %u", &j);
 printf("\nValue of j = %u", j);
 printf("\nValue of i = %d", i);
 printf("\nValue of i = %d", *(&i));
 printf("\nValue of i = %d", *j);
 return 0;
}
```

Pointers: pointerExample2.c

```
#include <stdio.h>
int main(){
 int i=3;
 int *j;
 j = &i;
 printf("\nAddress of i = %u", &i);
 printf("\nAddress of i = %u", j);
 printf("\nAddress of j = %u", &j);
 printf("\nValue of j = %u", j);
 printf("\nValue of i = %d", i);
 printf("\nValue of i = %d", *(&i));
 printf("\nValue of i = %d", *j);
 return 0;
}
```

Output:

```
Address of i = 2293532
Address of i = 2293532
Address of j = 2293528
Value of j = 2293532
Value of i = 3
Value of i = 3
Value of i = 3
```

Key Concepts Related to Pointers

- Declaring a pointer

```
int *myIntPtr;
int* myIntPtr;
```

- Getting the address of a variable

```
int age = 3;
myIntPtr = &age;
```

- Dereferencing a pointer

```
*myIntPtr = 5;
```

Note: We just changed the value of age!

Pointers Example 2: ptrExample.c

```
#include <stdio.h>
int main(){
    int myValue;
    int *myPtr;
    myValue = 15;
    myPtr = &myValue;
    printf("myValue is equal to : %d\n", myValue);
    *myPtr = 25;
    printf("myValue is equal to : %d\n", myValue);
}
```

Output:

```
myValue is equal to : 15
myValue is equal to : 25
```

Passing Address to Function: passValue3.c

```
#include <stdio.h>

void addUpdate(int *a, int *b){
  int c;
  c = *a + *b;
  printf("Addition is : %d\n",c);
  *a = c;
  *b = c;
}
int main(){
  int a, b;
  printf("Enter Any 2 Numbers : ");
  scanf("%d %d",&a,&b);
  printf("a is: %d, b is: %d\n", a, b);
  addUpdate(&a, &b);
  printf("a is: %d, b is: %d\n", a, b);
  return 0;
}
```

Notice the pointer

Notice &a, &b

Note: The values of a and b changed in addUpdate function .

Output of passValue3.c

- Output:

```
Enter Any 2 Numbers : 2 8
a is: 2, b is: 8
Addition is : 10
a is: 10, b is: 10
```

Pointers and Arrays

- The square-bracket array notation is a short cut to prevent you from having to do pointer arithmetic

```
char array[5];
array[2] = 12;
```

`array` is a pointer to **`array[0]`**

`array[2] = 12;` is therefore equivalent to

```
*(array+2) = 12;
```

Passing Array to Function

```
#include <stdio.h>
double averageVal(int *matrixA, int M){
    int i, sum =0;
    double average;
    for(i=0; i<M; i++){
        sum = matrixA[i] + sum;
    }
    average = sum/M;
    return average;
}
int main(){
  int N;
  double avg;
  int myMatrix[4] = {1, 2, 3, 4};
  N= 4;
  avg = averageVal(myMatrix, N);
  printf("\naverage is: %lf\n", avg);
  return 0;
}
```

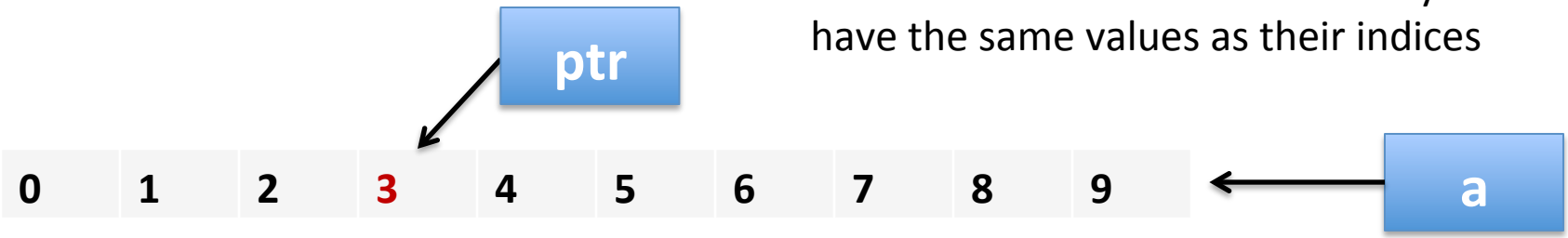
Pointer Arithmetic

- We have already seen
  - Pointers to variables
  - Correspondence between pointers and arrays

- Pointers can also be used to point to elements/cells of an array

```
int *ptr;
int a[10];
ptr = &a[3];
```

Assume that the elements of array **a** have the same values as their indices



Pointer Arithmetic

- Pointers can also be used to point to elements/cells of an array

```
int *ptr;
int a[10];
ptr = &a[3];
```

- **`*ptr`** will give us the value stored at **`a[3]`**

- What happens if 1 is added to the pointer **`ptr`**?

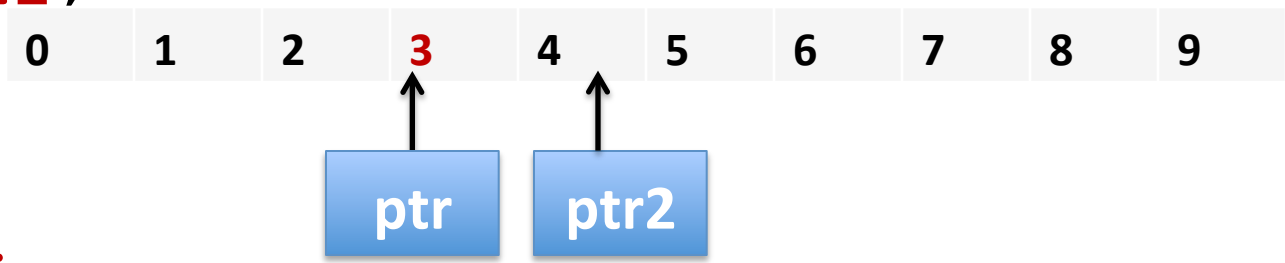
  `ptr + 1`

  The pointer will point to the next array element **`a[4]`**

Pointer Arithmetic

- Pointers can also be used to point to elements/cells of an array

```
int *ptr, *ptr2;
int a[10];
ptr = &a[3];
ptr2 = ptr +1;
```



- What happens if do the following?

```
*ptr2 = 100;
```

`a[4]` will be set to 100.

- We can achieve whatever we did through **`ptr2`** by using **`*ptr`** directly

```
*(ptr +1) = 100;
```

Pointer Arithmetic

- Note the usage of parentheses in the following:

```
*(ptr +1) = 100;
```

- If you do not add parentheses then "*****" operator (value of operator) will have higher precedence over "**+**":

`*(ptr +1)` will fetch the value at one further location from where **`ptr`** is pointing to.

`*ptr +1` will fetch the value pointed to by **`ptr`** and will add **`1`** to it.

Pointer Arithmetic

- What will the following do?

```
*(ptr + 6) = 68;
*(ptr - 2) = 40;
```

- Where will **ptr** point to after the following sequence of steps?

```
ptr = ptr + 1;
ptr  += 1;
ptr++;
```

Assume that **ptr** is pointing to `a[3]` .

Pointer Arithmetic

- An expressions like `*ptr++` will give access to the value pointed by `ptr` and will also increment it by 1.

- Assuming that `ptr` is pointing to `a[3]`, what does the following expression do?

`*++ptr`

- The above expression will increment the pointer and will then access the value it points to.

Journey of Pointer

- Write a program in which you declare a one-dimensional, integer array named `a` that can hold 10 elements
- Declare an integer pointer
- Initialize the array elements to the square of their index value

```
a[i] = i*i;
```

- Let the pointer point to the fourth element in the array – that is `a[3]`
- Print the following to trace the journey of the pointer:

```
*ptr,  *ptr+1, *(ptr+1),  *ptr++,  *(ptr++),
*++ptr, *(++ptr), *ptr
```

References

- C Programming Language, Brian Kernighan and Dennis Ritchie
- Let Us C, Yashavant Kanetkar
- C for Dummies, Dan Gookin
- http://cplusplus.com
- https://en.wikibooks.org/wiki/C_Programming/Strings