# ECE 220 Computer Systems & Programming

Lecture 11: Pointers and Arrays

February 19, 2019

ECE ILLINOIS

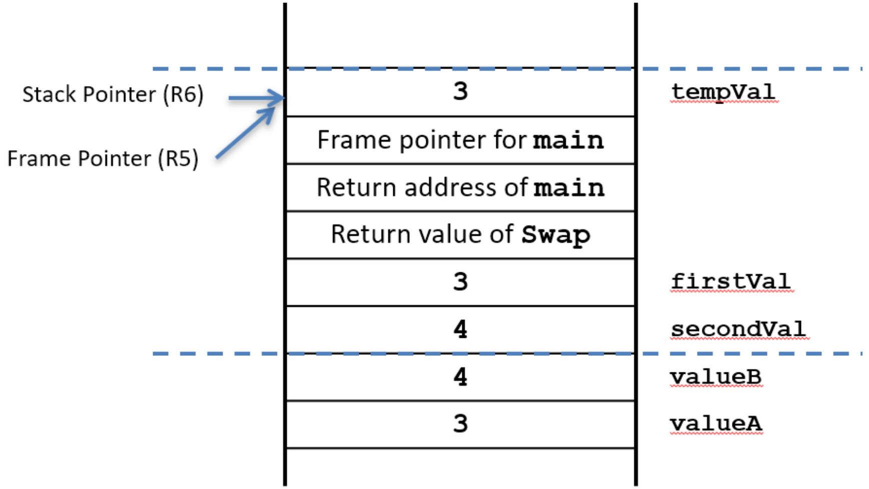
ILLINOIS

## Outline

- Chapter 16
- Key concepts
  - Passing by reference with pointers
  - Arrays basics

# Function **Swap**

```
#include<stdio.h>

void Swap(int firstVal, int secondVal);

int main()
{
        int valueA = 3;
        int valueB = 4;

1.      printf("%d %d\n”, valueA, valueB);
2.      Swap(valueA, valueB);
3.      printf("%d %d\n", valueA, valueB);
4.      return 0;
}

void Swap(int firstVal, int secondVal)
{
        int tempVal;

5.      tempVal = firstVal;
6.      firstVal = secondVal;
7.      secondVal = tempVal;
}
```



# Function NewSwap

```
#include<stdio.h>

void NewSwap(int *firstVal, int *secondVal);

int main()
{
        int valueA = 3;
        int valueB = 4;

1.      printf("%d %d\n”, valueA, valueB);
2.      NewSwap(&valueA, &valueB);
3.      printf("%d %d\n", valueA, valueB);
4.      return 0;
}

void NewSwap(int *firstVal, int *secondVal)
{
        int tempVal;

5.      tempVal = *firstVal;
6.      *firstVal = *secondVal;
7.      *secondVal = tempVal;
}
```

# Pointers

## Declaration

```
int *p;   /* p is a pointer to an int */
```

A pointer in C is always a pointer to a particular data type: `int*`, `double*`, `char*`, etc.

## Operators

| | |
|---|---|
| `*p` | -- returns the value pointed to by p |
| `&z` | -- returns the address of variable z |

## Example

```
int object;
int *ptr;

object = 4;
ptr = &object;
*ptr = *ptr + 1;
```

store the value 4 into the memory location associated with “object”

store the address of “object” into the memory location associated with ptr

read the contents of memory at the address stored in ptr

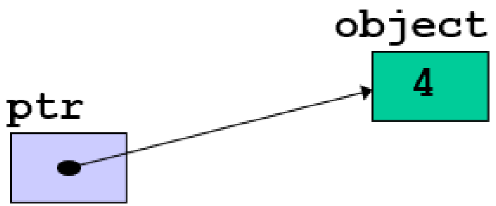
store the result into memory at the address stored in ptr

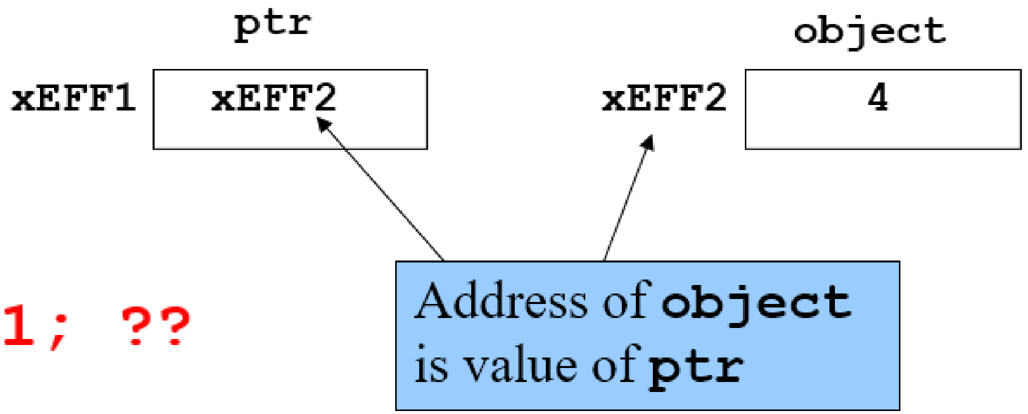
- **&** (address operator)

```
ptr = &object;
```

  - Returns address of operand

```
int object = 4;
int *ptr;
ptr = &object; //ptr gets address of object
ptr "points to" object
```



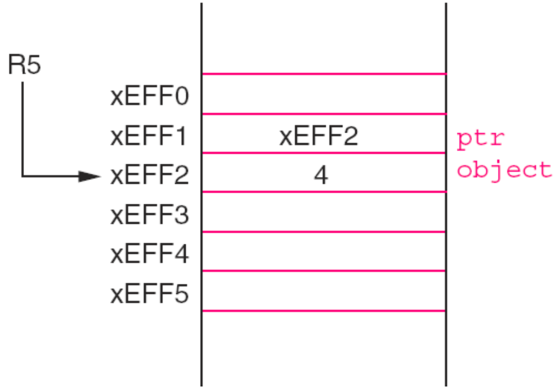


```
*ptr = *ptr + 1; ??
```

# Pointers in LC3

- The indirection operator '*'

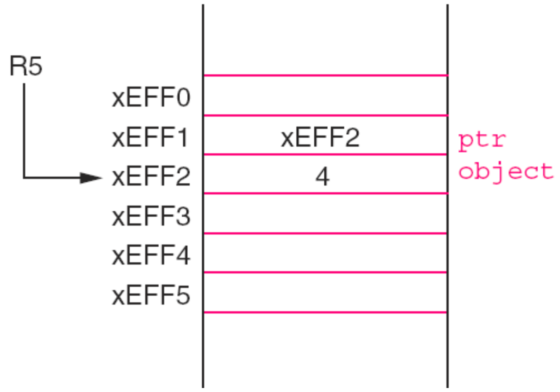
```
int object = 4;
int *ptr;
ptr = &object;
```



```
AND   R0, R0, #0    ;    Clear R0
ADD   R0, R0, #4    ;    R0 = 4
STR   R0, R5, #0    ;    Object = 4;

ADD   R0, R5, #0    ;    Generate memory address of object
STR   R0, R5, #-1   ;    Ptr = &object;
```

# Pointers in LC3

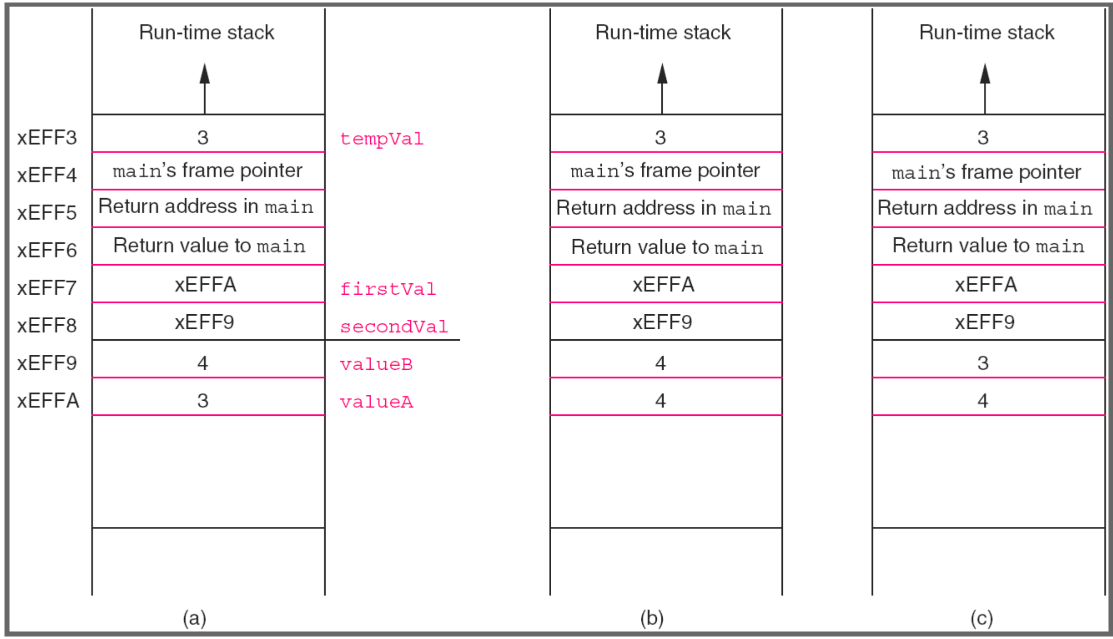
*ptr = *ptr + 1; ??



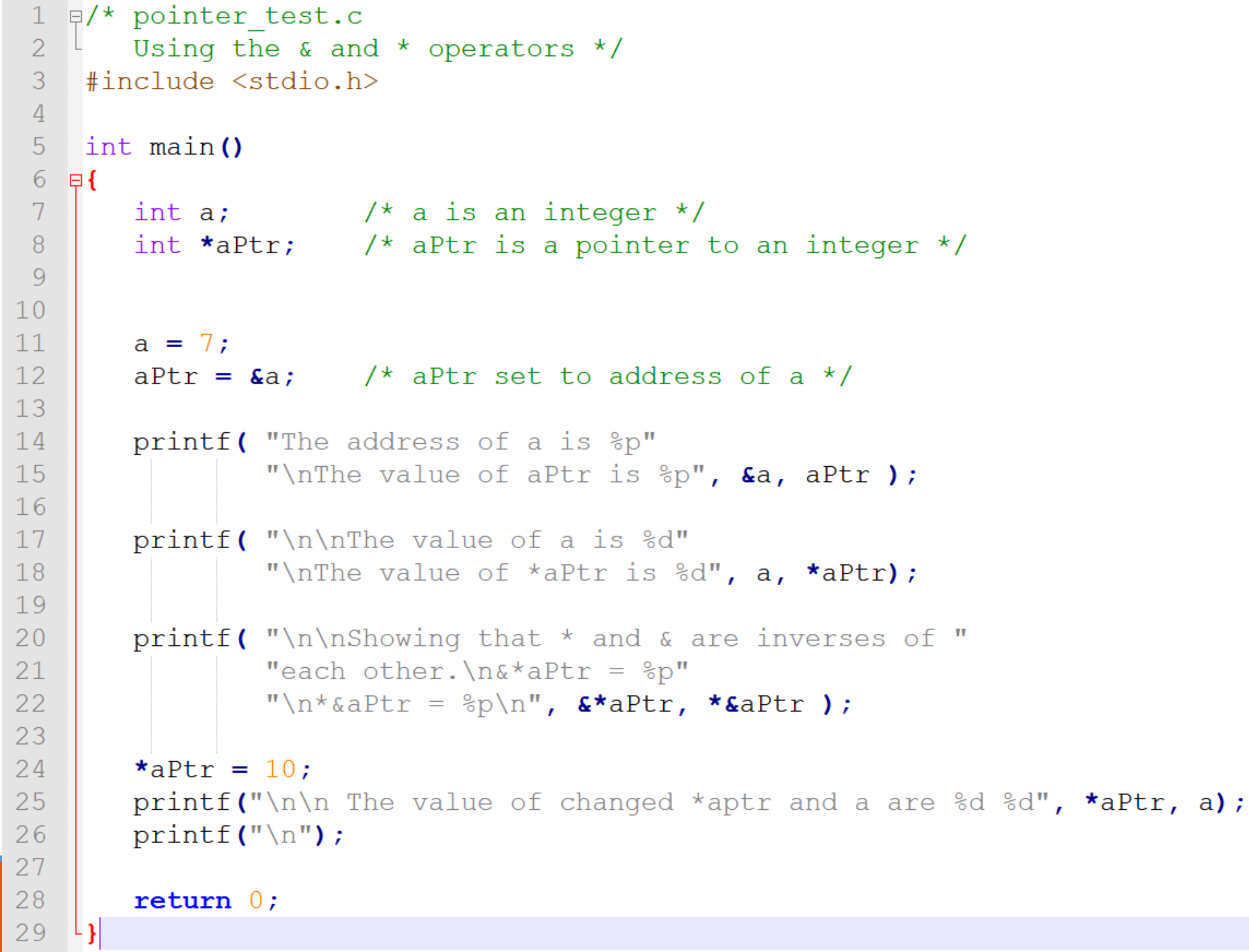
```
LDR   R0, R5, #-1    ;   R0 contains the value of ptr
LDR   R1, R0, #0     ;   R1 <- *ptr
ADD   R1, R1, #1     ;   *ptr + 1
STR   R1, R0, #0     ;   *ptr = *ptr + 1;
```

# Function NewSwap

```
#include<stdio.h>

void NewSwap(int *firstVal, int *secondVal);

int main()
{
        int valueA = 3;
        int valueB = 4;

1.      printf("%d %d\n”, valueA, valueB);
2.      NewSwap(&valueA, &valueB);
3.      printf("%d %d\n", valueA, valueB);
4.      return 0;
}

void NewSwap(int *firstVal, int *secondVal)
{
        int tempVal;

5.      tempVal = *firstVal;
6.      *firstVal = *secondVal;
7.      *secondVal = tempVal;
}
```

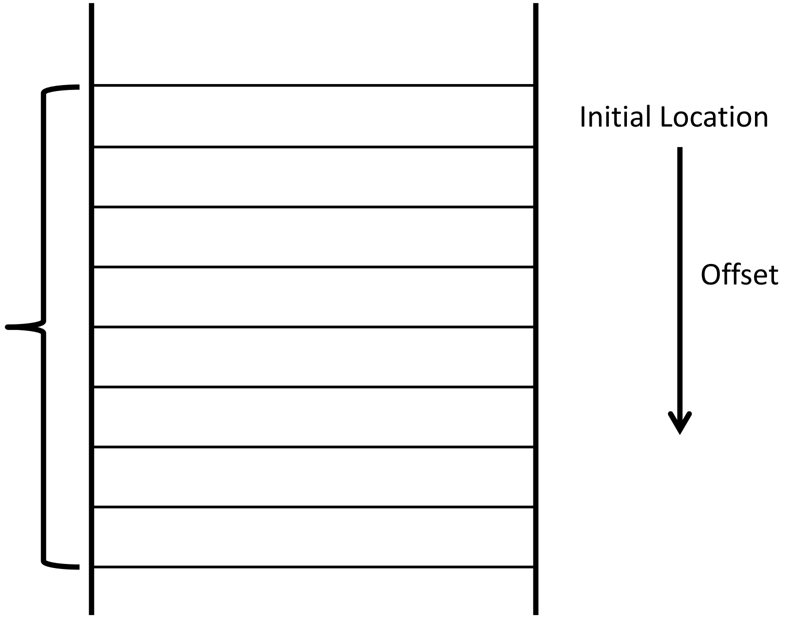


**Exercise:**



```c
/* pointer_test.c
   Using the & and * operators */
#include <stdio.h>

int main()
{
   int a;           /* a is an integer */
   int *aPtr;       /* aPtr is a pointer to an integer */


   a = 7;
   aPtr = &a;       /* aPtr set to address of a */

   printf( "The address of a is %p"
           "\nThe value of aPtr is %p", &a, aPtr );

   printf( "\n\nThe value of a is %d"
           "\nThe value of *aPtr is %d", a, *aPtr);

   printf( "\n\nShowing that * and & are inverses of "
           "each other.\n&*aPtr = %p"
           "\n*&aPtr = %p\n", &*aPtr, *&aPtr );

   *aPtr = 10;
   printf("\n\n The value of changed *aptr and a are %d %d", *aPtr, a);
   printf("\n");

   return 0;
}
```

# Arrays: Basic Concept



## Arrays: Basic Concept

**How do we allocate a group of memory locations?**

- **character string**
- **table of numbers**

**How about this?**

```
int num0;
int num1;
int num2;
int num3;
```

**Not too bad, but…**

- **what if there are 100 numbers?**
- **how do we write a loop to process each number?**

**Fortunately, C gives us a better way -- the *array*.**

```
int num[4];
```

**Declares a sequence of four integers, referenced by:**
`num[0]`, `num[1]`, `num[2]`, `num[3]`.

## Arrays: Syntax

### Declaration

```
type   variable[num_elements];
```

all array elements are of the same type

number of elements must be known at compile-time

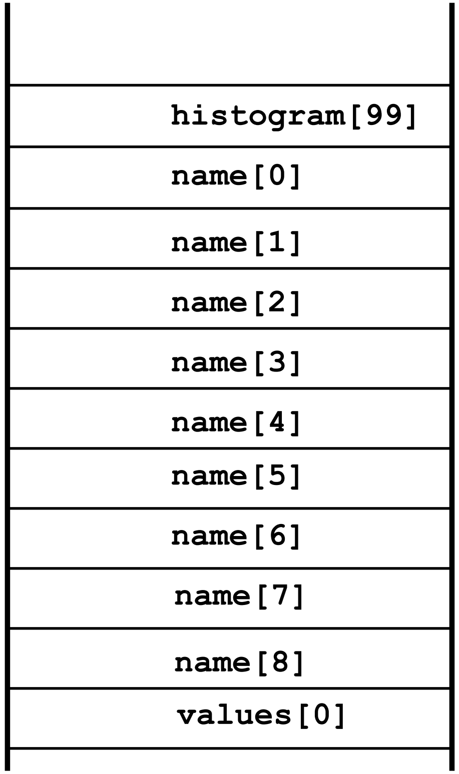
### Array Reference

```
variable[index];
```

i-th element of array (starting with zero); no limit checking at compile-time or run-time

# Arrays in C

```
int main()
{
    int histogram[100];
    char name[9];
    double values[1024];

    .
    .
    histogram[6] = histogram[6] + 1;

    i = 0;
    while (name[i] != '\0')
      i++;
    .
    .
}
```



## Arrays Example

- **Declaring and using Arrays**

```
int grid[10] = {0,1,2,3,4,5,6,7,8,9};
grid[6] = grid[3] + 1;
int i;
for(i=0;i<2;i++)
{
   grid[i+1] = grid[i] + 2;
}
```

## Passing Array as Arguments

**C passes arrays by reference**

- the address of the array (i.e., address of the first element) is written to the function's activation record
- otherwise, would have to copy each element
- `[]`indicate to the compiler that the corresponding parameter will be the base address of an array of the specified type.

```
int main(){
  int array[10];
  int result;
  result = average(array);
  return 0;
}

int average(int array[]);
```

# Array Concepts (Next Class)

- Bounds checking
- Assignment of arrays
- Passing Arrays as parameters in functions
- Pointers and Arrays