

# EECS 10: Computational Methods in Electrical and Computer Engineering

## Lecture 22

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering  
Electrical Engineering and Computer Science  
University of California, Irvine

## Lecture 22: Overview

- Pointers
  - Definition, initialization and assignment
  - Pointer dereferencing
  - Pointer arithmetic
    - Increment, decrement
  - Pointer comparison
  - String operations using pointers
    - Pointer and array type equivalence
    - Passing pointers to functions
    - Type qualifier `const`
  - Standard library functions
    - String operations defined in `string.h`

## Pointers

- *Pointers* are variables whose values are *addresses*
  - The “address-of” operator (&) returns a pointer!
- Pointer Definition
  - The unary \* operator indicates a pointer type in a definition

```
int x = 42; /* regular integer variable */
int *p;    /* pointer to an integer */
```

- Pointer initialization or assignment
  - A pointer may be set to the “address-of” another variable
  - A pointer may be set to 0 (points to no object)
  - A pointer may be set to **NULL** (points to “NULL” object)

```
p = &x; /* p points to x */
```

```
p = 0; /* p points to no object */
```

```
#include <stdio.h> /* defines NULL as 0 */
p = NULL; /* p points to no object */
```

EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

3

## Pointers

- Pointer Dereferencing
  - The unary \* operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>
int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */
```

**p**

0

**x**

42

EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

4

## Pointers

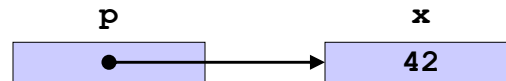
- Pointer Dereferencing
  - The unary \* operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>

int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */

p = &x; /* make p point to x */
printf("x is %d, content of p is %d\n", x, *p);
```

```
x is 42, content of p is 42
```



EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

5

## Pointers

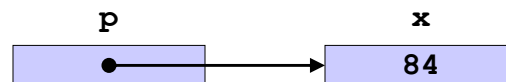
- Pointer Dereferencing
  - The unary \* operator dereferences a pointer to the value it points to (“content-of” operator)

```
#include <stdio.h>

int x = 42; /* regular integer variable */
int *p = NULL; /* pointer to an integer */

p = &x; /* make p point to x */
printf("x is %d, content of p is %d\n", x, *p);
*p = 2 * *p; /* multiply content of p by 2 */
printf("x is %d, content of p is %d\n", x, *p);
```

```
x is 42, content of p is 42
x is 84, content of p is 84
```



EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

6

## Pointers

- Pointer Dereferencing
  - The `->` operator dereferences a pointer to a structure to the content of a structure member

```

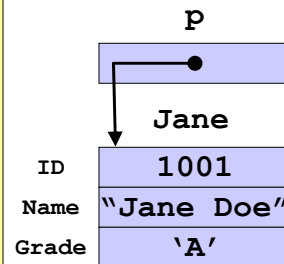
struct Student
{
    int ID;
    char Name[40];
    char Grade;
};

struct Student Jane =
{1001, "Jane Doe", 'A'};

struct Student *p = &Jane;

void PrintStudent(void)
{
    printf("ID:    %d\n", p->ID);
    printf("Name:   %s\n", p->Name);
    printf("Grade:  %c\n", p->Grade);
}

```



```

ID:    1001
Name:  Jane Doe
Grade: A

```

EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

7

## Pointers

- Pointer Arithmetic
  - Pointers pointing into arrays may be ...
    - ... incremented to point to the next array element
    - ... decremented to point to the previous array element

```

int x[5] = {10,20,30,40,50}; /* array of 5 integers */
int *p; /* pointer to integer */

p = &x[1]; /* point p to x[1] */
printf("%d, ", *p); /* print content of p */

```

```

20,

```

EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

8

## Pointers

- Pointer Arithmetic

- Pointers pointing into arrays may be ...

- ... incremented to point to the next array element
- ... decremented to point to the previous array element

```
int x[5] = {10,20,30,40,50}; /* array of 5 integers */
int *p; /* pointer to integer */

p = &x[1]; /* point p to x[1] */
printf("%d, ", *p); /* print content of p */
p++; /* increment p by 1 */
printf("%d, ", *p); /* print content of p */
```

20, 30,

## Pointers

- Pointer Arithmetic

- Pointers pointing into arrays may be ...

- ... incremented to point to the next array element
- ... decremented to point to the previous array element

```
int x[5] = {10,20,30,40,50}; /* array of 5 integers */
int *p; /* pointer to integer */

p = &x[1]; /* point p to x[1] */
printf("%d, ", *p); /* print content of p */
p++; /* increment p by 1 */
printf("%d, ", *p); /* print content of p */
p--; /* decrement p by 1 */
printf("%d, ", *p); /* print content of p */
```

20, 30, 20,

## Pointers

- Pointer Arithmetic

- Pointers pointing into arrays may be ...

- ... incremented to point to the next array element
- ... decremented to point to the previous array element

```
int x[5] = {10,20,30,40,50}; /* array of 5 integers */
int *p; /* pointer to integer */

p = &x[1]; /* point p to x[1] */
printf("%d, ", *p); /* print content of p */
p++; /* increment p by 1 */
printf("%d, ", *p); /* print content of p */
p--; /* decrement p by 1 */
printf("%d, ", *p); /* print content of p */
p += 2; /* increment p by 2 */
printf("%d, ", *p); /* print content of p */
```

20, 30, 20, 40,

## Pointers

- Pointer Comparison

- Pointers may be compared for equality

- operators == and != are useful to determine *identity*
- operators <, <=, >=, and > are *not* applicable

```
int x[5] = {10,20,10,20,10}; /* array of 5 integers */
int *p1, *p2; /* pointers to integer */

p1 = &x[1]; p2 = &x[3]; /* point to x[1], x[3] */

if (p1 == p2)
{ printf("p1 and p2 are identical!\n");
}
if (*p1 == *p2)
{ printf("Contents of p1 and p2 are the same!\n");
}
```

Contents of p1 and p2 are the same!

## Pointers

- Pointer Comparison

- Pointers may be compared for equality

- operators == and != are useful to determine *identity*
- operators <, <=, >=, and > are *not* applicable

```
int x[5] = {10,20,10,20,10}; /* array of 5 integers */
int *p1, *p2;                /* pointers to integer */

p1 = &x[1]; p2 = &x[3];      /* point to x[1], x[3] */
p1 += 2;                      /* increment p1 by 2 */
if (p1 == p2)
{ printf("p1 and p2 are identical!\n");
}
if (*p1 == *p2)
{ printf("Contents of p1 and p2 are the same!\n");
}
```

```
p1 and p2 are identical!
Contents of p1 and p2 are the same!
```

## Pointers

- String Operations using Pointers

- Example: String length

```
int Length(char *s)
{
    int l = 0;
    char *p = s;

    while(*p != 0)
    { p++;
      l++;
    }
    return l;
}
```

```
char s1[] = "ABC";
char s2[] = "Hello World!";

printf("Length of %s is %d\n",
       s1, Length(&s1[0]));
printf("Length of %s is %d\n",
       s2, Length(&s2[0]));
```

```
Length of ABC is 3
Length of Hello World! is 12
```

## Pointers

- String Operations using Pointers

- Example: String length

```
int Length(char *s)
{
    int l = 0;
    char *p = s;

    while(*p != 0)
    { p++;
      l++;
    }
    return l;
}
```

```
char s1[] = "ABC";
char s2[] = "Hello World!";

printf("Length of %s is %d\n",
       s1, Length(&s1[0]));
printf("Length of %s is %d\n",
       s2, Length(s2));
```

```
Length of ABC is 3
Length of Hello World! is 12
```

- Array and pointer types are equivalent

- `s2` is an array, but can be passed as a pointer argument
- Character array `s2` is same as character pointer `&s2[0]`

## Pointers

- String Operations using Pointers

- Example: String length

```
int Length(char *s)
{
    int l = 0;
    char *p = s;

    while(*p != 0)
    { p++;
      l++;
    }
    return l;
}
```

```
char s1[] = "ABC";
char *s2 = "Hello World!";

printf("Length of %s is %d\n",
       s1, Length(s1));
printf("Length of %s is %d\n",
       s2, Length(s2));
```

```
Length of ABC is 3
Length of Hello World! is 12
```

- Array and pointer types are equivalent

- `s1` is an array of characters, `s2` is a pointer to character
- Both `s1` and `s2` can be passed to character pointer `s`



## Pointers

- String Operations using Pointers

- Example: String length

```
int Length(char s[])
{
    int l = 0;
    char *p = s;

    while(*p != 0)
    { p++;
      l++;
    }
    return l;
}
```

```
char s1[] = "ABC";
char *s2 = "Hello World!";

printf("Length of %s is %d\n",
      s1, Length(s1));
printf("Length of %s is %d\n",
      s2, Length(s2));
```

```
Length of ABC is 3
Length of Hello World! is 12
```

- Array and pointer types are equivalent

- `s1` is an array of characters, `s2` is a pointer to character
- Both `s1` and `s2` can be passed to character array `s`

## Pointers

- String Operations using Pointers

- Example: String copy

```
void Copy(
    char *Dst,
    char *Src)
{
    do{
        *Dst = *Src;
        Dst++;
    } while(*Src++);
}
```

```
char s1[] = "ABC";
char s2[] = "Hello World!";

printf("s1 is %s, s2 is %s\n",
      s1, s2);

Copy(s2, s1);
printf("s1 is %s, s2 is %s\n",
      s1, s2);
```

```
s1 is ABC, s2 is Hello World!
s1 is ABC, s2 is ABC
```

- Passing pointers as arguments to functions

- Function can modify caller data by pointer dereferencing
- **Passing pointers = Pass by reference!**

## Pointers

- String Operations using Pointers

- Example: String copy

```
void Copy(
    char *Dst,
    const char *Src)
{
    do{
        *Dst = *Src;
        Dst++;
    } while(*Src++);
}
```

```
char s1[] = "ABC";
char s2[] = "Hello World!";

printf("s1 is %s, s2 is %s\n",
      s1, s2);

Copy(s2, s1);
printf("s1 is %s, s2 is %s\n",
      s1, s2);
```

```
s1 is ABC, s2 is Hello World!
s1 is ABC, s2 is ABC
```

- Passing pointers as arguments to functions

- Function can modify caller data by pointer dereferencing
- Type qualifier **const**:  
Modification by pointer dereferencing *not* allowed!

## Pointers

- String Operations using Pointers

- Example: String copy

```
void Copy(
    const char *Dst,
    const char *Src)
{
    do{
        *Dst = *Src;
        Dst++;
    } while(*Src++);
}
```

Error!  
Write access to  
const data!

```
char s1[] = "ABC";
char s2[] = "Hello World!";

printf("s1 is %s, s2 is %s\n",
      s1, s2);

Copy(s2, s1);
printf("s1 is %s, s2 is %s\n",
      s1, s2);
```

```
s1 is ABC, s2 is Hello World!
s1 is ABC, s2 is ABC
```

- Passing pointers as arguments to functions

- Function can modify caller data by pointer dereferencing
- Type qualifier **const**:  
Modification by pointer dereferencing *not* allowed!

## Standard Library Functions

- Functions declared in `string.h` (part 1/2)
  - `typedef unsigned int size_t;`
    - type definition for length of strings
  - `size_t strlen(const char *s);`
    - returns the length of string `s`
  - `int strcmp(const char *s1, const char *s2);`
    - alphabetically compares string `s1` with string `s2`
    - returns -1 / 0 / 1 for less-than / equal-to / greater-than
  - `int strncmp(const char *s1, const char *s2, size_t n);`
    - same as previous, but compares maximal `n` characters
  - `int strcasecmp(const char *s1, const char *s2);`
  - `int strncasecmp(const char *s1, const char *s2, size_t n);`
    - same as string comparisons above, but case-insensitive

EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

21

## Standard Library Functions

- Functions declared in `string.h` (part 2/2)
  - `char *strcpy(char *s1, const char *s2);`
    - copies string `s2` into string `s1`
  - `char *strncpy(char *s1, const char *s2, size_t n);`
    - copies maximal `n` characters of string `s2` into string `s1`
  - `char *strcat(char *s1, const char *s2);`
    - concatenates string `s2` to string `s1`
  - `char *strncat(char *s1, const char *s2, size_t n);`
    - concatenates maximal `n` characters of string `s2` to string `s1`
  - `char *strchr(const char *s, int c);`
    - returns a pointer to the first character `c` in string `s`, or `NULL` if not found
  - `char *strrchr(const char *s, int c);`
    - returns a pointer to the last character `c` in string `s`, or `NULL` if not found
  - `char *strstr(const char *s1, const char *s2);`
    - returns a pointer to the first appearance of `s2` in string `s1` (or `NULL`)

EECS10: Computational Methods in ECE, Lecture 22

(c) 2009 R. Doemer

22