7. Arrays and Strings
ENEE 140

Prof. Tudor Dumitraș
Associate Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

1

Today's Lecture

- Where we've been
  - Scalar data types (`int, long, float, double, char`)
  - Integer and floating point arithmetic
  - Basic control flow (`while` and `if`)
  - Functions
- Where we're going today
  - Vector data types: arrays, strings
  - *Defensive programming and* `assert()`
  - *Testing*
  - Project 1 Q&A
- Where we're going next
  - Project 1: partial implementation due on Friday
  - Midterm exam (after the Spring Break)
  - Complex programs

2

2

Scalar vs. Vector Data Types

- We've seen
  `char, int, long, float, double`
  - These are **scalar** data types: a variable holds a single value

- **Vector** data types: hold a series of scalar variables of the same type
  - Declaration must specify the **size N** of the array

```
int     a[10];      int array with N=10 elements
long    b[10];      long array with N=10 elements
float   c[10];      float array with N=10 elements
double  d[10];      double array with N=10 elements
char    e[10];      string with up to 9 characters (!)
```

  - Accessing array elements: **index** between **0** and **N-1**

```
a[0] = 0;           first element
a[9] = 0;           last element
```

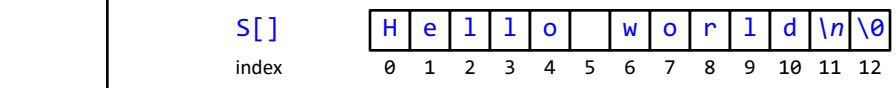
3

3

Strings

- Strings are character arrays, with some special rules
  - You can initialize strings using string literals (use double quotes)

`char s[] = "Hello world\n";` size of s[] is implicit



  - The character `'\0'` indicates the end of the string

`char s[10];` must account for '\0' => can only store 9 chars

  - You can read and write strings using scanf and printf
    - Use the `%s` format modifier

```
char s[] = "Hello world\n";
printf("The string is: %s", s);
```

4

4

Initialization vs. Assignment

- Arrays and strings can be initialized, but **can not be assigned**

```
char s1[] = "ENEE 140";          s1 is declared and initialized
char s2[10];                     s2 is declared but not initialized
int a[3] = {1, 2, 3}, b[3];      a initialized, b uninitialized
s2 = "ENEE 140";                 error! (cannot assign strings)
b = a;                           error! (cannot assign arrays)
```

- Instead, arrays and strings **can be copied**
  - String copy

```
#include <string.h>              needed for strncpy

strncpy(s2, "ENEE 140", 10);     must specify the size of s2[]
```

  - Array copy

```
for (i=0; i < 3; i++) {
  b[i] = a[i];                   copy a[] element-by-element
}
```

5

5

Reading Strings

- `scanf`: input string stops at whitespace or at the max field width

```
char s[10];
scanf("%9s", s);        specify field width 9 to allow for '\0' terminator
                        note: s instead of &s
```

- `fgets`: read whole line up to specified size – 1

```
fgets(s, 10, stdin);    stdin is the standard input stream
                        (more on this later)
```

  - The '\n' character will be included in s[]
  - `fgets()` returns NULL on EOF or error

- Read input line-by-line, until EOF is encountered

```
while (fgets(s, 10, stdin) != NULL) { … }
```

- Use a string as input source

```
sscanf(s, "%d", &i);        read integer i from string s[]
```

6

6

Writing Strings

- printf: use %s format specifier

```
char str[] = “world”;
printf(“Hello %s\n", str);
```

- fputs: print only the string

```
fputs(str, stdout);
```

stdout is the standard output stream

- Use a string as output:

```
sprintf(str, “%3d”, i);
```

write integer i into str[]

  - **Important: Must be careful not to exceed the size of str[]!**

7

7

Common Programming Mistakes

- Accessing or modifying elements outside the array bounds
  - Incorrect

```
int  a[10];
a[-1] = 0;
a[10] = 0;

char s[10];
scanf("%s", s);
```

index can be

can store up to

  - Correct

```
a[i] = 0;
scanf("%9s", s);
```

where 0 <= i < 10

specify field width

- **This is one of the most common security vulnerabilities in software!!** (more about this in ENEE 457)

8

Defensive Programming

- Good programming practice:
  - Think about relationships among the variables in your program
  - Determine conditions (e.g. a == b+1) that must be true at various steps, if your program is correct
  - Force the program to stop when these conditions are violated, then test the program with a variety of inputs to make sure that this doesn't happen
  - This approach is called "defensive programming"

- Assert: a tool for defensive programming

```
#include <assert.h>
assert(condition);
```

exits the program if *condition* is false

  - Use assert() liberally
  - Assertions allow you to diagnose mistakes in your program
  - They also make your assumptions clear to other programmers who will read your code

9

9

Defensive Programming – Example

- Use defensive programming to prevent common mistakes related to arrays and strings

```
#include <assert.h>

int a[10];
assert(i>=0 && i<10);
a[i]=0;
```

exits before accessing index out of bounds

- Turn off all assertions at compile time

```
gcc –NDEBUG myprogram.c
```

10

10

String Functions

- Convenient operations on strings

```
#include <string.h>
```

`strlen(s);`	length of s
`strncpy(dst, src, n);`	copy up to n characters from src to dst
`strncat(dst, src, n);`	concatenate dst and src
`strncmp(s1, s2, n);`	compare s1 and s2

- Common programming mistake
  - Using `strcpy`, `strcat`, `strcmp`, etc.
  - These functions do not allow you to specify the size of the destination string
  - **Always use the strn* functions instead of the str* functions!**

11

11

The `sizeof` Operator for Vector Data Types

- Yields the number of bytes required to store the array or string
  - Array dimension x size of base type

```
char a[10];
int b[10];
sizeof(a)        10
sizeof(b)        40
```

- **Important: string size vs. string length**

```
char c[10] = "enee140";
sizeof(c)        10
strlen(c)        7
```

12

12

Function Parameters

- For scalar types (e.g. `int`, `float`), we've seen:
  - Modifying the arguments inside the function **does not** affect the original variables
  - The function operates on a **copy of the variable**

```
int b, a = 2;
my_function(a);
b = a + 1;
```

a is still 2, regardless of what happens in the function

- Vector types (e.g. `array`, `string`):
  - Modifying the elements of the array inside the function **does** change the original variable
  - The function operates on the **original array**

```
void
empty_string(char s[])
{
 s[0] = '\0';
}

char str[] = "Hello world";
empty_string(str);
printf("%s", str);
```

function with string parameter

empty string "" is printed

13

13

Return Values

- The value returned from a function cannot be a vector type
  - You cannot return `int[]` or `char[]`
  - You must return a scalar type, e.g. `int` or `char`
  - You can also write a function that does not return anything (using `void`)
- Common programming practice
  - To perform operations that produce a **scalar** data type, write a function that **returns the value** you are trying to compute
  - To manipulate a **vector** data type, write a function that **takes as parameter the string or array that will hold the result** of the operation

14

14

Testing

- Complex programs are more likely to have bugs
- It is important to test these programs thoroughly, with a broad range of inputs
  - Create several sets of input values (**test cases**)
  - Think about **corner cases**
- Good programming practice: write test cases **before** writing the program
  - This helps you clarify what the program should do
- **Debugging is not enough** for writing correct programs
  - You must also create **rigorous tests**

16

16

Review of Lecture

- What did we learn?

17

17

Next Steps

- Next time
  - **Midterm exam (after Spring Break)**
    - Midterm review recording on web page
- Next lecture
  - Complex programs
- Assignments for this week
  - **Project 1**: partial implementation due on **3/14**
  - No homework — **study for the midterm**
- Assignments for week after the midterm
  - Homework: `lab07.pdf`, due on Friday at 11:59 pm
  - Read **K&R Chapters 4.3, 4.4, 4.5, 4.6, 4.8, 4.9, 4.11**
  - Weekly challenge: `trim_strings.c`
  - **Quiz 5**, due on Sunday at 11:59 pm

18

18