Week 1.1

Introduction

School of Information Technology and Electrical Engineering
The University of Queensland

CSSE 2310 7231

Welcome

- **CSSE2310 / CSSE7321**
  - Computer Systems Principles and Programming
- Teaching Staff
  - Dr Joel Fenwick
  - A/Prof Peter Sutton
  - Tutors (Adam, Nathaniel, Pat, Richard, Simon, Thomas)
- Rule 0: If you have questions, then ask.

What's This Course All About?

- Exposure to UNIX operating system
  - Shell commands
- Underlying Principles of
  - Operating Systems
  - Computer Networks
- Systems Programming C

- You will become more effective programmers and system designers by having knowledge of the underlying systems

Resources

- Course website
  - http://courses.itee.uq.edu.au/csse2310/2012s1
  - Lecture Slides
    - Usually posted just in advance of lectures
  - Pracs
  - – Programming problems and exercises
- Notices
  - Distributed via newsgroup, subject site
  - by email if urgent

Communication

- **Newsgroup:** uq.itee.csse2310 – Best method to communicate with staff and other students.
  - [less good] MyNewsgroups on my.uq
  - Reader software: eg mozilla thunderbird
- Joel:
  - Email: joelfenwick@uq.edu.au
  - Anonymous feedback link on the subject page.
    - If I don't know who you are, then I can't respond.
    - Very little on the net is truly anonymous

Course Profile

- Describes
  - The course in detail
  - What you can expect
  - What we expect of you
- **You should obtain and read the course profile**
- Now for *some* of the details …

Assumed Background

- You must know something about programming

- The more comfortable you are with programming in general, the easier you will find this course
- You should also have …
  … Some knowledge of computer systems

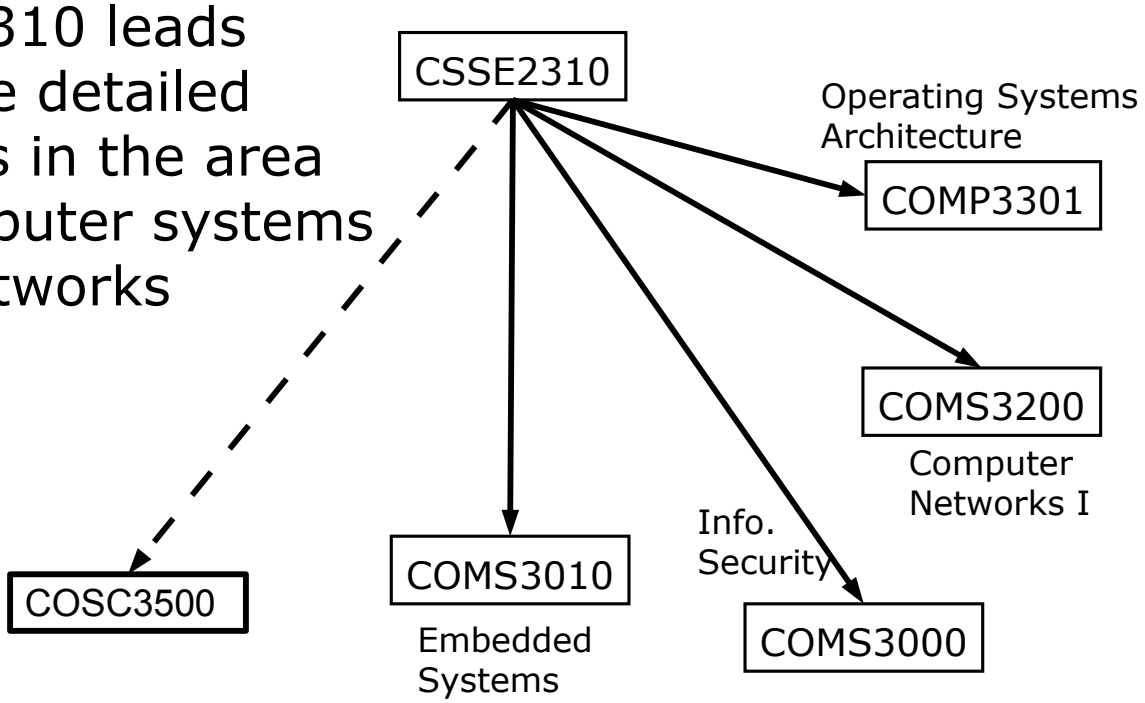
  … Knowledge of binary representations (2's complement etc)

  … Knowledge of binary operations (AND, OR, XOR, …)

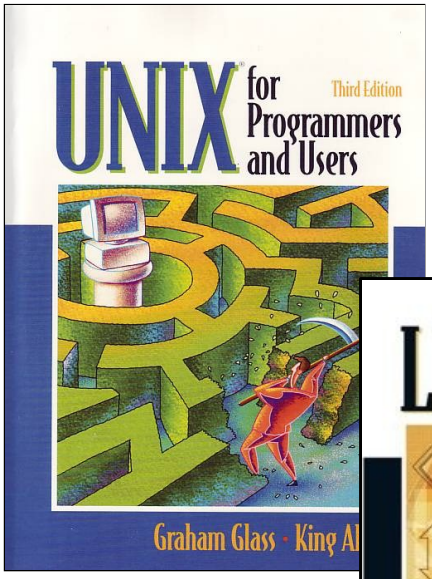
  … Ideally, some prior exposure to C

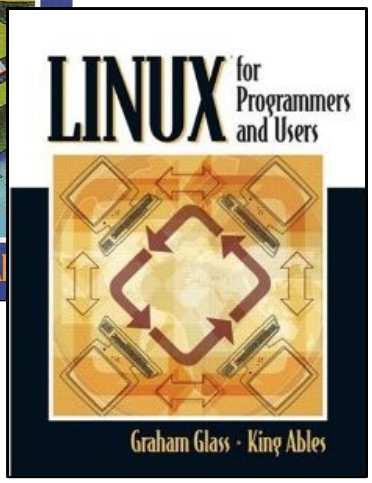
Other Courses

- CSSE2310 leads to more detailed courses in the area of computer systems and networks



Textbooks



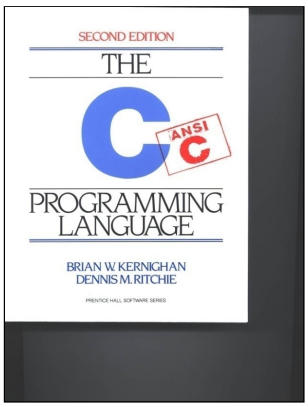


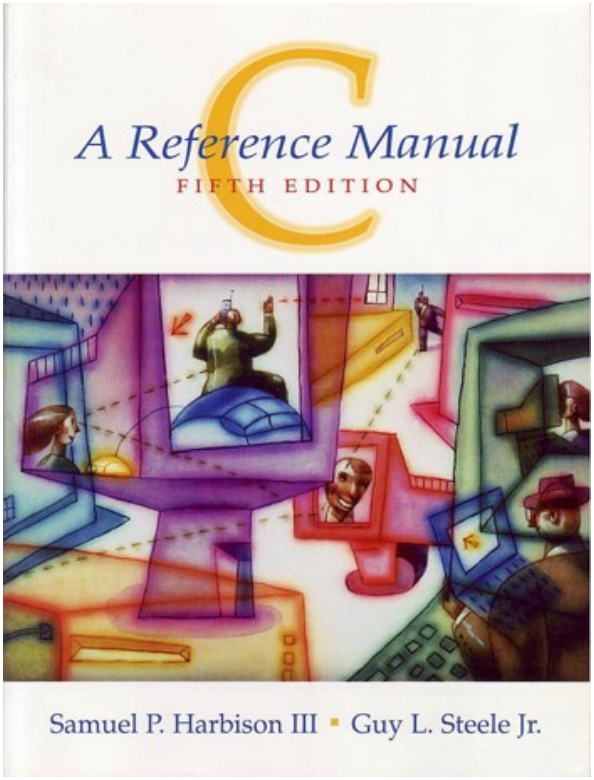
- Glass & Ables
  - UNIX for Programmers and Users
  - Linux for Programmers and Users
- Covers most aspects of the course

Textbooks (cont.)

- Harbison and Steele -
  - C: A Reference Manual (5th edition)
  - Highly recommended as a **reference** on C
- Kernighan and Ritchie
  - The C Programming Language (2nd ed, 1988)
  - Does not cover C99

OR





Assessment

- **Assignments** (100 marks total)
  - Four Assignments
    - Equal weight (25 marks) but not equal difficulty
    - A1 – Simple C Programming
    - A2 – debugging
    - A3 and A4 – UNIX systems programming in C
- **Exams** (100 marks total)
  - Mid-semester exam (in Friday lecture, week 7)
    - Multiple choice, open book
  - Final exam
    - Written answers, open book
  - Overall exam mark is better of
    - 30% mid-semester + 70% final
    - 15% mid-semester + 85% final
  - Exams cover theory and programming

Grade determination

- Final mark (out of 100) determined as **geometric mean** of assignment and exam marks (and then rounded to nearest integer)

$$Final_{mark} = \sqrt{Assignment_{mark} \times Exam_{mark}}$$

- No minimum requirements on exam or assignment marks.
- Grade determined from final mark

7 = 85 to 100	4 = 50 to 64
6 = 75 to 84	3 = 45 to 49
5 = 65 to 74	2 = 20 to 44

  - CSSE7321 has different cutoffs

Late Submissions

- Assignments are all due electronically
  - 1,3,4 using subversion
  - Submission by **11pm** on due date
  - 20% per 24hrs (or part thereof) late penalty
  - No submissions accepted more than 96 hours after the deadline under any circumstances.
- **Read course profile for the fine print!**

Plagiarism and Collusion

- All assignments are individual
  - All submitted code must be your work
  - Using code provided on CSSE2310 website is acceptable
  - Use of any other published code is unacceptable
- **ALL submitted code will be subject to plagiarism and collusion detection**
- Don't copy or look at code from other students or allow your code to be copied or seen – this is cheating
  - Misconduct proceedings will be initiated if plagiarism and/or collusion is found
- You are encouraged to discuss assignments but this should not include sharing code

Plagiarism and Collusion (cont.)

- Assessment can serve (at least) two purposes:
  - Feedback to you on your learning
  - Measuring your performance for the purpose of generating a grade
- Plagiarism and/or collusion compromises both of these

What to do help you learn

- Lectures
- Tutorials (enroll if you have not already)
- Assignments (Not just testing what you've learned elsewhere.)
  - You will gain a better understanding of by doing the assignments.
  - Lectures do not give detailed instructions for assignments.
  - We don't discuss some problems until people ask about them.
- Private study

What to expect in pracs

- Exercises early in weeks.
  - Eg intro to unix
- Teaching in some weeks.
  - Eg make
- Work on assignments, get help.
- You may attend as many tutorials as you wish but enrolled students have priority.

What to expect in lectures

- Stand and stretch breaks half way through each hour (approximately)
- 10 minute break in the middle of Tuesday lecture
- Stories!
- Some practical examples, tool demos, explanations
- Take notes!
  - Lecture slides don't capture everything
- Lectures will generally cover higher level concepts (except for weeks 2-4)
- Mid-semester exam in Friday lecture slot in week 7

Things I may do during lectures

Ask questions

- "Why?" - you may need to justify answers.
- I don't expect people to be able to answer all questions immediately.
- May need to move quickly to give someone else a chance.
  - Dealing with some answers may require material we haven't covered.

Things I may do during lectures

Employ comical exaggeration

- Concepts are abstract
- Computers are fast
- Hard to differentiate between good and bad solutions

What we expect from you…

- **Attendance** at lectures
  - You may be disadvantaged if you don't attend
- **Seek help if you're having trouble**
  - Don't leave it too late
- **Hard work**
  - Ask students from previous years.
- **Feedback and ideas** (anonymous if you like)
  - What can we improve?
    - Especially if some aspect of the course is causing you distress.
  - What do you want to learn about?
    - Course is pretty full so no major changes.

Facilities

- Pracs in **78-{108, 208, 116, 336}**
  - PC lab, from which you can remotely access LINUX server
  - After hours access available
    - You'll need an access card – see the Faculty office
  - Login using UQ password
- Server: **moss.labs.eait.uq.edu.au**
  - Runs Linux
  - Access from lab PCs possible, via
    - ssh (command line)
    - X-window (graphical)
  - Remote access possible
    - ssh to **moss.labs.eait.uq.edu.au**
    - See http://studenthelp.itee.uq.edu.au/remote/

Using your own hardware (optional!)

- Connect to moss via ssh (putty)
- Work on your own computer. At your own risk. Always test on moss! If it does not work on moss it does not work!
- If your computer is running:
  - Linux – Make sure you have gcc, make and svn installed.
  - MacOSX – You will need to install the X-Code from your OS cd/app store.
  - Windows – consider installing linux.

Linux at home

- If you haven't done so already. This is a good opportunity to try linux on your own hardware.
  - Lots of people to answer questions.
  - Can work without connecting to moss.
    - Always test on moss.
- While we can answer questions we do not provide support for install problems.
  - We probably won't debug on your hardware.
  - If it eats your pets and destroys your computer – not our fault!

Install options

- Virtual machine: A program simulates a whole computer on which you can install and run an OS.
  - VirtualBox, vmware, parallels
- Dual boot: Choose between a number of OS at boot time. (Need to reboot to switch).
  - Wubi – windows installer for Ubuntu
  - Debian, Ubuntu, many others
- Use your isp's mirrors where possible

CSSE 2310 7231

Week 1.2

C-Introduction

Pracs

- Enrol in two sessions (one P session and one C session) per week.
- Only P sessions run in week 1.
- Over the next week or so [this will take more than tute time]:
  - Unix tutorial exercise
  - C programming tutorials
  - C programming exercises

Lecture Outline

- UNIX editors
- Building C programs
- C Programming Language
  - Basic structure of a program
  - Quick overview of some features
  - Arrays
  - Pointers
  - Structures
  - Preprocessor

UNIX Editors

- It is highly recommended that you learn to use a UNIX text editor
- Two popular editors, suitable for writing programs are
  - **vi** (or vim – "vi improved")
  - **emacs**
- See pages 57 to 75 of Glass & Ables for a brief introduction to both
- More details, including links to tutorials are on the course website

Building C programs

- C program files are typically named
  <name>.c
  i.e., lowercase .c extension
- Programs are *compiled* and *linked* to produce an *executable*
- **gcc** command can be used for both compilation and linking
  - **gcc** (used to be GNU C Compiler, now **GNU Compiler Collection**) is a free compiler collection – available for many systems

Hello World

Compilation and Linking

- Explanation in class

Use of gcc

- **Compilation** (production of object code)
  - `gcc -c` *name*`.c`
  - -c argument means compile but do not link
  - Example above will produce file *name*`.o`
- **Compilation and Linking in one step**
  - `gcc` *name*`.c`
    - Links with standard C library and produces executable named a.out
  - `gcc -o` *executable-name* *name*`.c`
    - -o argument specifies the name of the output file

Use of gcc (cont.)

- **Linking**
  - `gcc -o` *executable-name* *name*`.o`
- Can give multiple filenames as arguments, e.g.
  - `gcc -o` *executable-name* *name1*`.c` *name2*`.c` *name3*`.o`
    - Compiles and links as required
- Sometimes need to link with the maths library (-lm) if program uses maths functions
  - `gcc -o` *executable-name* *name1*`.c` *name2*`.c` … `-lm`

Why have separate compilation/linking?

- Large programs are made up of multiple source files
- If change one file, shouldn't have to recompile all the others, just
  - recompile the one that changed
  - link the object files to produced an executable
- Recompiling everything can be a slow process
- The **make** command (and Makefiles) provide an automated mechanism to only recompile files that change
  - More details later

C Programming Language

- In this course we expect you to…
  - be able to write C programs from scratch
  - understand the meaning of C programs
  - be able to modify C programs
  - understand how C programs use memory
- Lectures can't teach programming
- You'll need to practice

C Program – Basic Structure

- Main function name must be **`main`**
  - This function is executed when program starts
- Blocks of code enclosed by braces { }
- C statements must end with a semicolon ;
- C statements are case sensitive
  - **`variable`** is not the same as **`Variable`**
- Comments are within /* … */
  - // accepted by newer compilers (C99)
    - Comment is from // to end of line
    - Initially, we'll use /* … */ only

Basic Structure (cont.)

- C program consists of
  - Declarations
  - Function definitions
- Function definitions have
  - Variable declarations
  - Statements

Declaring Variables

- Declaration
  - **`type-name variable-name, variable-name …;`**
- Examples
  - **`char c;`**
  - **`int day;`**
  - **`unsigned int count;`**
  - **`float expense, income;`**
  - **`double pi;`**

Single byte

Integers (size is machine dependent) Integers can be unsigned or signed (two's complement).

Two single precision floating point numbers

A double precision floating point number

- char, int, float, double are among the data types supported by C
- C does not have a separate **`boolean`** type (Java does) C99 has **bool**. You will need to `#include<stdbool.h>`

Function Definitions

What type is returned by the function, e.g. **`int`** or **`void`**

Name of the function, e.g. **`main`**

Argument (parameter) declaration, e.g. **`int a, float b`**

```
return-type function-name (arg declaration)
{
    variable-declarations;
    …
    statements;
}
```

Variables used ONLY within the function

The code which actually does stuff

Function Example

```
/*
Return the average of two integers
   (result will be rounded towards 0)
*/
int average(int a, int b) {
 int avge;

 avge = (a+b)/2 ;

 return avge;
}
```

Statements

These are **expressions.** Outer expression is a statement.

The function **returns** a result when finished

C Constants

- **Character constants**
  - Use single quotes, e.g. 'a', 'b', '1' etc
  - Some special characters – **backslash escaped**
    - '\n' = newline, '\'' = single quote, '\t' = tab, '\\' = backslash
- **String constants**
  - Use double quotes (can include backslash escapes) e.g. "abc \n \" hello\t"
- **Integer constants**
  - Decimal – e.g. 3 , -27 , 65535 , +5
  - Hexadecimal (leading 0x), e.g. 0x5F , 0xFFFF , 0xDEADBEEF
  - Octal (leading 0), e.g. 0377 (= 255 decimal)

C Constants

- **Boolean** (can always use integers)
  - [c99] bool, true, false
- **Floating point constants**
  - Include decimal point (.) and/or "e" for exponent
  - Examples: 3.1416 , -7. , 6.02e23 , -5.2e-2
  - Note 7 is an integer, 7. is floating point

Some Operators

- **Binary operators**

+	addition
–	subtraction
*	multiplication
/	division
%	remainder (integer)
>	greater than
>=	greater than or equal
==	equal
!=	not equal
<	less than
<=	less than or equals
&	bitwise AND
|	bitwise OR
^	bitwise XOR
&&	logical AND
||	logical OR

- **Unary operators**

!	logical not
~	one's complement (invert)
–	two's complement (negate)
++	increment (prefix or postfix)
– –	decrement (prefix or postfix)

More Operators: Bit-shifting and assignment

- `a << b` means a shifted left by b bits
- `a >> b` means a shifted right by b bits
  - What bits are shifted in from the left depends on whether a is signed or not. Do not rely on this.
- `a = b`means a is assigned the value of b
- `a += b` is shorthand for `a=a+b`
- Similarly `-=, *=, /=, %=, &=, |=, ^=, <<=, >>=`
- Examples

`1 << 5` is 1 * 25 = 32

`3 << 4` is 3 * 24 = 48

`a += 1` same as ++a

Postfix/Prefix Increment and Decrement

- Example:

```
int a,b,c,d,e;
a = 4;
b = a++;   /* b=a;   a=a+1; */
c = --a;  /* a=a-1; c=a;    */
d = ++a;  /* a=a+1; d=a;    */
e = a--;  /* e=a;   a=a-1; */
```

- After these statements, values are
  a=4, b=4, c=4, d=5, e=5

Postfix – change happens after the value used

Prefix – change happens before the value used

Operator Precedence

- Consider a + b * c
- C has strict operator precedence to disambiguate expressions like the above
- Above expression means a + (b * c)
- Some operators associate right to left, e.g.
  ~ ++ a means ~ (++ a)
- Most associate left to right:
  a - b - c means (a - b) - c not a - (b - c)

Operator Precedence and Associativity

Operators		Associativity
( ) [ ] -> .		Left to right
! ~ + - ++ -- & *	(unary versions)	Right to left
* / %		Left to right
+ -		Left to right
<< >>		Left to right
< <= > >=		Left to right
== !=		Left to right
&	(bitwise and)	Left to right
^	(bitwise xor)	Left to right
|	(bitwise or)	Left to right
&&	(logical and)	Left to right
||	(logical or)	Left to right
?:		Right to left
= *= /= %= += -= &= ^= |= <<= >>=	(assignment)	Right to left
,		Left to right

Increasing Precedence

Exercise (1)

- What's the result of this code?

You have 1½ minutes

Control Statements

- **`if (`*`expression`*`) `*`stmt`*** **`else `*`stmt`***
  - **`else`** clause is optional
- **`while (`*`expression`*`) `*`stmt`***
- **`do `*`stmt`*** **`while (`*`expression`*`)`**
- **`for(`*`init_expr; test_expr; end_expr`*`) `*`stmt`***

- Note on expressions:
  - C interprets any 0 value as false, anything else as true
  - (Java has a specific boolean type)
- *stmt* can be replaced by multiple statements enclosed in braces { }

Exercise (2)

- What's the result of this code?

You have 1 minute

For Loop Equivalent Code

```
for (init_expr; test_expr; iter_expr) statement1;
```

is equivalent to:

```
init_expr;
while (test_expr) {
    statement1;
    iter_expr;
}
```

Statement can be replaced by multiple statements enclosed in braces

- Any or all of the expressions can be empty
- Can use comma to separate multiple expressions:

```
for (i=0, j=0; i<10; i++, j+= 4)
```

Exercise (3)

- What's the result of this code?

You have 2 minutes

Function Return Values

- If no return type is given, C assumes **int**
- Where no return value is desired, the keyword **void** can and should be used
- It is an error to return the wrong type
- Good idea to prototype a function before it is used
  - Especially if used before being defined, or defined in another file
  - Header files (.h files) contain prototypes for library functions
- A prototype is like a call to a procedure, but appears outside any procedure
- Has no procedure body

Function Prototypes

```
int get_voltage(void);      /* Prototype for 1st proc */
void disp_voltage(int voltage);   /* Another prototype */

main() {                    /* main does not need a prototype */
v = get_voltage();
disp_voltage(v);
}

int get_voltage(void) {     /* Actual body for 1st */
    return inp(...);        /* procedure */
}
void disp_voltage(int voltage) {
    printf(...);
}
```


Where do variables live?

```
int a;
float b;

unsigned int max(unsigned int n1,
 unsigned int n2,
    unsigned int n3)
{
 int max;

 max=n1;
 if(n2 > max) max=n2;
 if(n3 > max) {
    max=n3;
 }
 return max;
}
```

"global" variables are allocated fixed addresses in memory

function variables are allocated memory every time the function is called. Memory is reclaimed at end of function.

Arrays

- Declaring an array

  ***type variable-name*[*size*];**

  - Examples:

```
char message[16];

int values[10];
```

- Accessing elements within an array

  ***variable-name*[*index*]**

  - index = 0 … size-1 (called zero-based indexing)
  - Examples:

```
message[0] = 'c';    values[9] = values[8]++;
```


Strings

- A string in C is an array of characters
  - End of string indicated by null character

Figure to be drawn in class

Arrays in Memory

- [To be presented in class]

Array Initialisation

- Arrays can be initialised at declaration, e.g.
  - `int values[9] = {3, 1, 4, 1, 5};`
    - if variable is global (static) – remaining elements initialised to 0
    - if variable is local (automatic) – remaining elements are uninitialised
- Size can be omitted if array is initialised, e.g.
  - `int a[] = {2,3,5,7};`
    - length is 4 in this case

Initialising String Arrays

- [To be presented in class]

Pointers

- C has concept of pointers
- Pointer declaration
  - ***type * variable-name;***
  - *variable-name* is a pointer to something of given *type*
    - How? – pointer variables store memory addresses
- Example:

```
char a, b;

char *ptr;


ptr = &a;

b = *ptr;
```

Can write these on one line:
`char a,b,*ptr;`

& is address-of operator – creates a pointer

* is indirection operator – returns value pointed to

- Figures to be drawn in class

Pointers and Arrays

- Array name can be treated as a pointer to the first element
  - i.e. address of first element
- Example:

```
int a[10];
int *ptr;


/* following statements are same */
ptr = a;
ptr = &a[0];
```

Operations on Pointers

- Addition/subtraction operations on pointers work in multiples of the size of the object being pointed to
- Example

```
int a[10];
int *ptr;

/* following statements are same */
ptr = a+5;
ptr = &a[5];
```


Traversing an Array

- Two examples of clearing an array
- Using **index**:

```
float a[10];
int index;
for(index=0; index<10; index++) {
    a[index] = 0.0;
}
```

Adding one to an array pointer makes it point to next element in array

- Using **pointer**:

```
float a[10], *ptr;
for(ptr=a; ptr < a+10; ptr++) {
    *ptr = 0.0;
}
```

Example Function

- Copying a string – can use index or pointer
- [One version to be presented in class, try writing the other yourself]

Example Function

Function Arguments

- Arguments passed to functions are copied (**passed by value**)
- Changes made within function don't affect original arguments
- Example:

```
void swap(int n1, int n2) {
   int tmp;
   tmp = n1;
   n1 = n2;
   n2 = tmp;
}
void main() {
   int a,b;
   a = 2;
   b = 3;
   swap(a,b);
   … /* nothing has happened */
}
```

This doesn't apply when arrays are passed to functions – since only a pointer to the array is passed.

Pointers and Functions

- If pass pointers as an argument to function, CAN change value that is pointed to (called **passing by reference**)
  - (The pointer is copied - not the value pointed to)
- Example:

```
void swap(int *n1, int *n2) {
   int tmp;
   tmp = *n1;
   *n1 = *n2;
   *n2 = tmp;
}
void main() {
   int a,b;
   a = 2;
   b = 3;
   swap(&a,&b);
   … /* a and b will be swapped */
}
```

Structures

- Like a class or record – groups several elements (called members or components) together:

```
/* Structure definition */
struct Time {
    int hour;     /* 0 - 23 */
    int minute;   /* 0 - 59 */
    int second;   /* 0 - 59 */
};
struct Time time; /* variable decl. */
```

This is the type

This is the variable name

- Members can be accessed using . (selection) operator

```
time.hour = 11;
minutes = time.hour*60 + time.minute;
```


typedef

- Can define new names for types
  - Often used with structures, but can be used for any type

```
typedef struct S {
    int a;
    int b;
} s1type;
```

S can be omitted from definition (have to use `s1type` name)

  - `s1type` is exactly the same as `struct S`

```
typedef int boolean;
```

  - Defines "boolean" to be a synonym for int

Structures and Pointers

- Pointers can point to structures
- Indirection operator ->
- [Code examples to be given in class]

Structures and Pointers

- Pointers can point to structures
- Indirection operator ->
- [Code examples to be given in class]

Things To Do This Week

- Learn a UNIX text editor
  - vi
  - Emacs
  - nano
- Learn C
  - Do C programming tutorials
  - Work on C programming exercises