Introduction to Pointers

15-123

Effective Programming in C and Unix

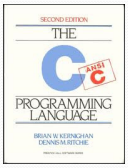
Announcements

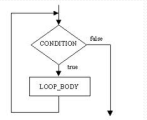
- SL2 is due Thursday 1/20 midnight
- Complete the Academic Honesty Form in class
- All code downloads are from
  - /afs/andrew/course/15/123/download
- All code handin's are at
  - /afs/andrew/course/15/123/handin
- All SL/lab feedback are given at
  - /afs/andrew/course/15/123/handback

Review

Topics so far

Bit Pattern	Decimal Value
0000 0000	0
0000 0001	1
1111 1111	-1
0000 0011	3
1111 1101	-3
0001 1111	31
1110 0001	-31





How data is read

RAM

File in hard drive

scanf /fscanf

```
#include <stdio.h>
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
```

- How do we read a file that contains one integer per line?
- How do we read a file that contains two ints per line?
- How do we read a file that contains an int char and a string?
- What happens if scanf("%d", x) (where x is an int) is executed?

Question

- What is the purpose of these function?

```
int pow(int x, int y){
  if (y==0) return 1;
  else
    return x*pow(x,y-1);
}

/* what does this function returns? */
int foo(int x, int y){
  int r = 1;
  while (y > 0) {
    if (y%2) r=r*x;
    x = x*x;
    y = y/2;
  }
  return r;
}
```

When will these functions fail?

What is the bug in the code?

```
int insertInOrder(int A[], int value){
  int j=next-1;
  while (j>=0 && A[j]>value)
    {A[j+1] = A[j];j--;}
  A[j] = value;
  next++;
  return EXIT_SUCCESS;
}
```

next is the index of the next available location in the array. For example if array has 5 elements then next is 5

Efficiency in operations

- What is the relation between
  - x = x + 1
  - x += 1
  - x++ or ++x
- Semantics

eg : int i =1;
while (i++ < 10) {statement} ➔ how many times statement executed?

eg : int i =1;
while (++i < 10) {statement} ➔ how many times statement executed?

A pointer to a character

- denoted by: char*
- char* ➔ reads "address of a character" (a.k.a String)

G	U	N	A	'\0'

- char* s ➔ means s is a pointer to character
  ➔ or address of a character
- char* s ➔➔ char *s ➔➔ char * s
  - Are all equivalent
  - Does not matter where the * is placed

Strings

- A String is an array characters
  - char word[n];
- Strings are mutable
  - word[0] = 'a';
- Strings ends with '\0' the null character (IMPORTANT)
- Reading a String
  - fscanf(fp, "%s", word);
- Comparing Strings
  - strcmp(s1, s2) > 0 or = 0 or < 0 (need <string.h>)

Strings ctd..

- char word[n];
  - word is the name of the array
  - word is also the starting address of word array
  - word is also a char*, that is a pointer to a character
  - word is a const pointer to the array
    - Is it possible to say: word = w1 if w1 is another string?
  - How do we copy strings ?
    - strcpy(s1, s2)

Important String functions

```
#include <string.h>

int strcmp(const char *s1, const char *s2);

int strncmp(const char *s1, const char *s2, size_t n);

char *strcpy(char *dest, const char *src);

char *strncpy(char *dest, const char *src, size_t n);
```

Arrays

- Syntax
  - <type> name_of_array[size_of_array];
    - Eg: int A[n]
  - <type>[ ] name_of_array;
    - Eg: int[] A;
- Semantics
  - A contiguous block of memory that holds <size_of_array> units of <type> variables
  - Name of the array is a "pointer" to the block
- Boundaries
  - Indices vary from 0,1,... size-1 (why?)
  - C does not check array boundaries

2D Arrays

- Syntax
  - <type> A[rows][cols];
    - char A[10000][32];
  - <type>[][] A;
    - char[][] A;
- Semantics
  - A contiguous block of memory to hold (rows*cols) values of <type>
- Memory representation
  - Since memory is 1D, 2D arrays are represented as a block long block of a 1D array
  - How does it then know the segmentation of columns?

2D Arrays ctd..

- Conceptually

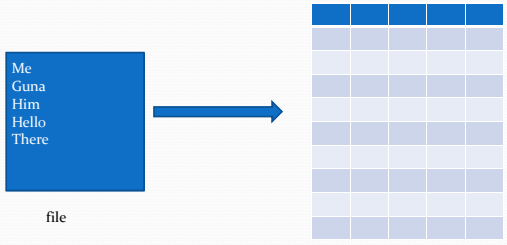
12	34	56
57	78	90

- Actually

12	34	56	57	78	90

- Hence column dimension is important
  - int foo(char A[][32], ...)
- Suppose you are given n(rows) by m(cols) int array A as a 1D array of m*n ints. What expression represent the (i,j)-th element of the array?

2D array of char's



file

Array in RAM

Read file

```
/*
   Function: readFile
   Purpose: To read a file of words into a 2D array of chars, sort if flag is true
   Input:  infile - a name of a file of words one word per line. Max word length is 31
           wordcount - an address of an int variable to hold the size of the file
           list - a 2D array of chars. Fixed row (182000) and column size (32)
           Flag - if true array will be sorted (calls insertInOrder for each word)
   Output: return 0 for success, 1 otherwise
   Called by: main
*/
int readFile(char* infile,  char list[][MAX_WORD_LENGTH], int flag){


  return 0;
}
```

Insert in order (strings)

```
/* Function: insertInOrder
   Purpose: This function inserts a word (in order) into the array of words
   Input: list is a 2D array of chars
          word is the word to be inserted in order
   output: return 0 is word can be inserted. 1 otherwise
   Algorithm: Find the place to insert the word as shifting other words to the right
   Called by: readFile
*/
int insertInOrder(char list[][MAX_WORD_LENGTH], int lastIndex,  char* word){


  return 0;
}
```

Write to file

```
/* Function: writeToFile
   purpose: This function writes the 2D array to a file
   input: outfile - name of a file to write
          wordcount - how many words to write to the file
          list - 2D array of words
   output: return 0 for success and 1 for failure
   Called by: main
*/
int writeToFile(char* outfile,  char list[][MAX_WORD_LENGTH]){

  return 0;
}
```

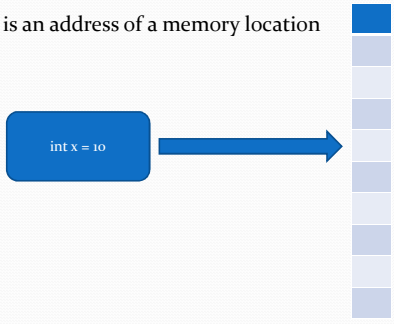
New material

Learning Objectives

- Understand how pointers work
- Understand how to access memory using pointers
- Understand pointer arithmetic
- Understand relation between arrays and pointers
- Understand the dangers of indirect memory access

Definition of a pointer

- A pointer is an address of a memory location



Pointer variables

- <type>* variable_name;
  - int* ptr; char* ptr;
- Initializing
  - int* ptr = &x ;
- Dereferencing
  - *ptr = x;

Pointer Arithmetic

- **<type>* ptr;**
- **ptr** ➔ address of a memory location
- **ptr + 1** ➔ address of the next <type> variable
- **ptr1 – ptr2** ➔ number of type variables between ptr1 and ptr2

Accessing Memory using pointers

- Suppose an int is defined as ➔ int x = 10
- How do we print the address of x?
- How do we access the first byte of x?
- How do we access the second byte of x?
-
- How do we access the next 4 bytes after x? What are the dangers?

Arrays

- Definition
  - int A[10]
  - char* A[10]
  - int* A[10]
- Array Memory allocation
  - Allocates a Contiguous block of memory
- The name of the array A is a constant pointer to the first element of the array
  - int A[10];
  - printf("%x", A);

Array as a pointer

Calculate the addresses of each element

Understanding Arrays

- A – address of the first element of the array
- A + i = address of A[i] = &A[i]

Dynamic Arrays

- int* A = (int*)malloc(n*sizeof(int));
  for (i=0; i<n; i++)
  A[i] = i;
- Resizing an array

C programming

- include <stdio.h> in all your programs
- Declare functions and variables before using them
- increment and decrement with ++ and – operators.
- Use x += 5 instead of x = x + 5
- A string is an array of characters ending with a '\0".
- Don't ever forget the null character.
- Array of size n has indices from 0 to n-1. Although C will allow you to access A[n] it is very dangerous.
- A character can be represented by an integer (ASCII value) and can be used as such.
- The unary operator & produces an address
- The unary operator * dereference a pointer
- Arguments to functions are always passed by value. But the argument can be an address of just a value
- For efficiency, pointers can be passed to or return from a function.
- Logical false is zero and anything else is true
- You can do things like for(;;) or while(i++) for program efficiency and writability
- Use /* .. */ instead of //
- Always compile your program with –ansi –pedantic –Wall flags

Coding Examples