# Lecture 4:
# Last of C Programming

# Logistics

- Programming Assignment 1 has been posted
  - Due July 13th 11:55pm
- Make sure to post questions on Piazza when possible
  - Piazza main forum for discussion
  - Others might have the same questions
  - Sign up for Piazza if you haven't already (via Sakai->Piazza)
- Keep looking at programming examples and try them out
  - On Sakai "Resources"->"Lecture_X"->"programs"

# Outline

- Live C examples
- Common Bugs
- Debugging with GDB
- Programming Assignment I Short Overview

# Some Live Coding

# Reading from a file

# Linked List

# 2D Array

# Other Stuff

# Common Bugs

# Classic Memory Bugs

- Memory management is one of the biggest differences between C and Java
- Let’s go over some bugs that might afflict you

# Bug #1: scanf() bug

- What is the issue with the following code?

```
scanf(“%d”, val);
```

- scanf() stores input using pointer arguments
- Need to use an ampersand (&) to address variable val

```
scanf(“%d”, &val);
```

# Bug #2: Memory Allocation And Use

- What is the issue with the following code?

```
/* return y = Ax */
int *matvec(int **A, int *x) {
    int *y = (int *) malloc(N*sizeof(int));
    int i, j;

    for (i=0; i<N; i++)
        for (j=0; j<N; j++)
            y[i] += A[i][j]*x[j];

    return y;
}
```

- Assumes that heap data is initialized to zero

# Bug #3: Overwriting Memory

- What is the issue with the following code?

```
int **p;
p = (int **)malloc(N*sizeof(int));

for (i=0; i<N; i++) {
   p[i] = (int*)malloc(M*sizeof(int));
}
```

- Allocating the possibly wrong sized object

# Bug #4:

- What is the issue with the following code?

```
int **p;
p = (int **)malloc(N*sizeof(int *));
for (i=0; i<=N; i++) {
   p[i] = (int *)malloc(M*sizeof(int));
}
```

- Off by one error, goes past array bounds

# Bug #5: Pointers

- What is the issue with the following code?

```
int *search(int *p, int val) {
    while (*p && *p != val)
        p += sizeof(int);
    return p;
}
```

- Misunderstanding pointer arithmetic

# Bug #6

- What is the issue with the following code?

```
int *foo () {
    int val;
    return &val;
}
```

- Forgetting that local variables disappear when a function returns

# Bug #7

- What is the issue with the following code?

```
x = malloc(N*sizeof(int));
...
free(x);
y = malloc(M*sizeof(int));
...
free(x);
```

- Freeing the same memory multiple times

# Bug #8

- What is the issue with the following code?

```
x = malloc(N*sizeof(int));
...
free(x);
...
y = malloc(M*sizeof(int));
for (i=0; i<M; i++)
    y[i] = x[i]++;
```

- Referencing freed memory

# Bug #9

- What is the issue with the following code?

```
void foo() {
    int *x = malloc(N*sizeof(int));
    ...
    return;
}
```

- Failing to free dynamically allocated memory
    - Memory leak
    - Will slowly eat up memory over time

# Bug #10

- What is the issue with the following code?

```
struct list {
    int val;
    struct list *next;
};

void foo() {
    struct list *head = (struct list*) malloc(sizeof(struct list));
    head->val = 0;
    head->next = NULL;
    <create and manipulate the rest of the list>
    ...
    free(head);
    return;
}
```

- Freeing only part of a data structure

# Debugging Programs With GDB

# GDB

- GNU Debugger
- Debugger for programming languages like C
- I use print statements, shouldn’t that be enough?
  - Sometimes it’s a hassle
- How to start start debugging using GDB:
  - First compile your program
    - Make sure to compile with ‘-g’ flag to compile with symbols
    - $ gcc –g –o program program.c
  - Run program with GDB
    - $ gdb ./program

# Common GDB Commands

- quit – quit GDB
- run – run program
- break – set breakpoint within program
  - break <function> - break before start of function
  - break <line #>   - break before particular line of code
- list – list lines around current point in program
- step – steps to next line (or into function)
- next – goes to next line (does not step into function)
- finish – continue until function returns
- continue – continue program execution

# Common GDB Commands (cont.)

- print – prints a variable
  - print <var> - prints variable
  - print/x <var> - prints in hexidemical format
  - https://sourceware.org/gdb/current/onlinedocs/gdb/Output-Formats.html
- watch – breaks if expression is changed
  - watch x – break if variable x is changed
- info – prints information about certain things
  - info break – prints information about breakpoints
  - Info variables – prints variables declared outside of functions
- backtrace – shows trace of function frames

# More about GDB

- If you want to learn more about GDB look into:
  - Tutorial: https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf
  - Tutorial: https://www.tutorialspoint.com/gnu_debugger/index.htm
  - Manual: https://ftp.gnu.org/old-gnu/Manuals/gdb/html_node/gdb_toc.html

# Programming Assignment 1: Brief Overview

# First: Array Sorting

- Sorts an input array
- Input
  - First line specifies number of array elements
  - Second line specifies the array
- Output
  - Odds in ascending order, Evens in descending order
- Example:

Input: [1 8 12 3 5 8 2 4 -5]

Sorted Array: [-5 1 3 5 12 8 8 4 2]

# Second: Hash Table

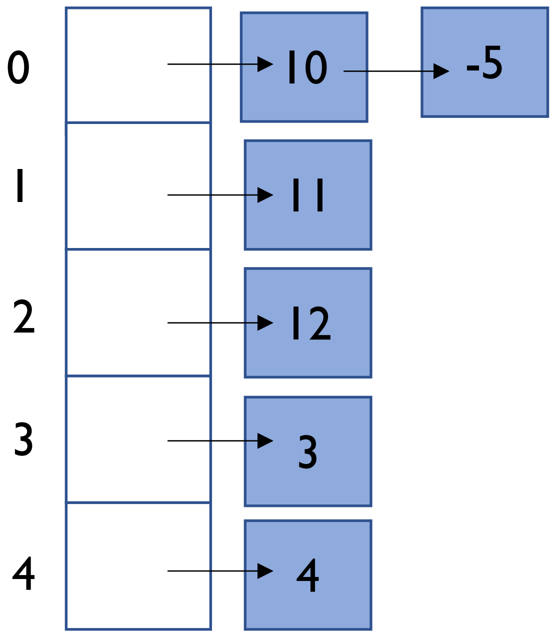
- Hash Table with 10,000 buckets
- Given hash table operations, manipulate hash table
  - i <value> : insert into hash table
    - prints “inserted” if value successfully inserted into hash table
    - prints “duplicate” if value already exists in hash table
  - s <value> : search hash table
    - prints “present” if value is in hash table
    - prints “absent” if value is NOT in hash table
- Hash is <value> modulo <# of buckets>
- Don’t forget about negative values
  - A negative value might yield you a negative hash value
  - Ex. -5 % 2 = -1, there is no such index as -1

# Second: Hash Table (Example)

- Example with 5 buckets

| example_file.txt | output |
|---|---|
| • i 10 | • inserted |
| • i 11 | • inserted |
| • i 3 | • inserted |
| • i 4 | • inserted |
| • i 12 | • inserted |
| • s 10 | • present |
| • i 10 | • duplicate |
| • s 4 | • present |
| • i -5 | • inserted |
| • s 5 | • absent |

Hash Table



# Third: Bit Function

- Take a value X and perform bit operations on it
- Operations:
  - set <n> <v> : Set the nth bit of X to v and print the value
  - get <n> <v> : Get the value of the nth bit and print it (eg.1 or 0)
  - comp <n> <v> : Modify the nth bit to its complement (eg. 1 if 0 or 0 if 1) and print the resulting X value
- Input
  - First line specifies the initial value X
  - Rest of lines specifies the bit operations to carry out
- Only use bitwise and assignment operators, no arithmetic
- Example:

| example_file.txt | state | output |
|---|---|---|
| • 5 | • 5 (0101) | • <none> |
| • get 0 0 | • 0101 | • 1 |
| • comp 0 0 | • 0101 -> 0100 = 4 | • 4 |
| • set 1 1 | • 0100 -> 0110 = 6 | • 6 |

# Fourth: One Shot Learning

- Carry out One-Shot Learning in C to predict the price of an unknown house
- Input:
  - File of training data with prices and attributes for a house in each row
  - File of attributes unknown houses
- Output:
  - The predicted prices for each of the unknown houses

# The Problem

- Someone out there is pricing houses based on several attributes
  - Ex. # of bedrooms, total size of house, # of baths, etc.
  - $x_1, x_2, x_3, x_4, \ldots.$
- All these attributes contribute to the final house price
- However all attributes may not contribute to the final price the same way
  - Not all attributes have the same weight on the price
- Would look more like the following:
  - $price = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \ldots$
  - Each attribute $x_i$ has some weight $w_i$
  - Note: $x_0$ is implicitly 1 as $w_0$ would basically serve as a constant

# The Problem

- Given some houses we do know the attributes and prices of, can we predict what the potential price of an unknown house?
- Example:
  - Recall: $price = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + \ldots$

| House Price | Price Constant | # bedrooms | # sq ft | # bathrooms |
|---|---|---|---|---|
| 350,000 | 1 | 4 | 3000 | 1 |
| 200,000 | 1 | 2 | 2000 | 1 |
| 577,000 | 1 | 5 | 8000 | 3 |
| 400,000 | 1 | 3 | 5000 | 2 |
| 300,000 | 1 | 3 | 4000 | 2 |
| ? | 1 | 3 | 5000 | 3 |

| House Price | | Price Constant | | # bedrooms | | # sq ft | | # bathrooms |
|---|---|---|---|---|---|---|---|---|
| 350,000 | = | $w_0$ 1 | + | $w_1$4 | + | $w_2$3000 | + | $w_3$ 1 |
| 200,000 | = | $w_0$ 1 | + | $w_1$ 2 | + | $w_2$ 2000 | + | $w_3$ 1 |
| 577,000 | = | $w_0$ 1 | + | $w_1$ 5 | + | $w_2$ 8000 | + | $w_3$ 3 |
| 400,000 | = | $w_0$ 1 | + | $w_1$ 3 | + | $w_2$ 5000 | + | $w_3$ 2 |
| 300,000 | = | $w_0$ 1 | + | $w_1$ 3 | + | $w_2$ 4000 | + | $w_3$ 2 |
| ? | = | $w_0$ 1 | + | $w_1$ 3 | + | $w_2$ 5000 | + | $w_3$ 3 |

- Can we learn weights $w_0,$ $w_1,$ $w_{2,}$ $and$ $w_3$ to predict the unknown house price?

# The Problem Generalized

- Lets solve the weights with what we know

| House Price | Price Constant | # bedrooms | # sq ft | # bathrooms |
|---|---|---|---|---|
| 350,000 | 1 | 4 | 3000 | 1 |
| 200,000 | 1 | 2 | 2000 | 1 |
| 577,000 | 1 | 5 | 8000 | 3 |
| 400,000 | 1 | 3 | 5000 | 2 |
| 300,000 | 1 | 3 | 4000 | 2 |

- Represent attributes and prices as matrices
- Let matrix X be a matrix of all the attributes we know
  - Each row corresponds to the attributes of a house
  - The ith column corresponds to the ith attribute
- Let W be a vector of all weights
- Let Y be the house prices

$$\underset{X}{\begin{bmatrix} 1 & 4 & 3000 & 1 \\ 1 & 2 & 2000 & 1 \\ 1 & 5 & 8000 & 3 \\ 1 & 3 & 5000 & 2 \\ 1 & 3 & 4000 & 2 \end{bmatrix}} \quad \underset{W}{\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}} \quad = \quad \underset{Y}{\begin{bmatrix} 350{,}000 \\ 200{,}000 \\ 577{,}000 \\ 400{,}000 \\ 300{,}000 \end{bmatrix}}$$

# Solution

- We have the equation $XW = Y$
- We know $X$ and $Y$
- We need to solve with for $W$
- We can do this using the following equation:
  - $W = (X^T X)^{-1} X^T Y$
- Method of Least Squares
  - Do not worry to much about the details
  - If interested look here: https://en.wikipedia.org/wiki/Linear_least_squares
- All you need to know is that it will approximate $W$
- Once we know $W$ we can use it to predict the price of any house

# Things you need to know

- Solving for the weights:
    - $W = (X^T X)^{-1} X^T Y$
- Predicting Price of Unknown House:
    - $Y = XW$
- Matrix Multiplication
    - https://www.mathsisfun.com/algebra/matrix-multiplying.html
    - https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:multiplying-matrices-by-matrices/v/matrix-multiplication-intro
- Transposing A Matrix
    - https://mathinsight.org/matrix_transpose
    - https://www.khanacademy.org/math/linear-algebra/matrix-transformations/matrix-transpose/v/linear-algebra-transpose-of-a-matrix
- Inverting a Matrix
    - https://www.mathsisfun.com/algebra/matrix-inverse-row-operations-gauss-jordan.html
    - https://www.khanacademy.org/math/algebra-home/alg-matrices/alg-determinants-and-inverses-of-large-matrices/v/inverting-matrices-part-3