CS 162
Intro to Computer Science II

Lecture 3

1D & 2D arrays

1/19/24

Oregon State University

Odds and Ends

- Office Hours location: KEC 1130
- Demo Location: KEC 1087 (starts week 4)
  - If you already signed up, feel free to reschedule or demo it early
  - One demo per assignment, -10 pts if missing your demo

- Due dates extended:
  - Lab 1 checkoff: Friday next week (Jan 26) for full credits
  - Design 1 + Quiz 1: this Sunday midnight (Jan 21)
  - Assignment 1: next Sunday midnight (Jan 28) (Demo due: Feb 9)

Additional Resources:

- random number generation:
  - Slides 7-8: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/slides/Lecture3.pdf
  - Code demo: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/demo/week3/rand.cpp
  - rand(): https://cplusplus.com/reference/cstdlib/rand/?kw=rand

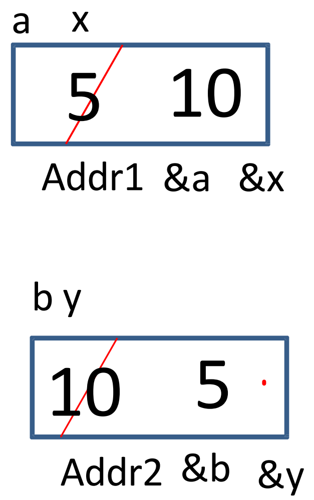
- Error handling:
  - Slides 3-9: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/slides/Lecture11.pdf
  - Code demo: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/demo/week8/error.cpp (note, you may use atoi() or stoi() instead)
  - stoi: https://cplusplus.com/reference/string/stoi/

Additional Resources:

- 1D array
  - Slides 6-12: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/slides/Lecture13.pdf
  - Code demo: https://classes.engr.oregonstate.edu/engr/winter2023/engr103-010/demo/week9/array.cpp

C++ Pass by Reference

```
void swap(int &, int &);
int main() {
   int a=5, b=10;
   swap(a, b);
   cout << “a: ” << a << “b: ” << b;
}
void swap(int &x, int &y) {
   int temp = x;
   x = y;
   y = temp;
}
```



Lecture Topics:

- 1D & 2D static arrays

1D static Arrays

- An array is a contiguous block of memory holding values of the same data type
- Static Arrays: created on the stack and are of a fixed size, during compiling time
  - 1-dimensional static array: `int stack_array[10];` type size
    - You can initialize an array at the same time as you declare it:

    ```
    int array[] = {1,2,3,4,5,6,7,8,9,10};
    ```

    Note: you can omit the size if you initialize the array when you declare it
    - Array name: stores the starting address of the array
    - i.e., array == &array == &array[0]
    - Conceptually, the array above looks like this:

Array index	0	1	2	3	4	5	6	7	8	9
Value	1	2	3	4	5	6	7	8	9	10

Passing a 1-D Array to functions

```
int main() {
  int array[5];
  …
  pass_1darray(array);
  …
}
void pass_1darray(int *a) {
  cout << “Array at zero: ” << a[0] << endl;
}
```

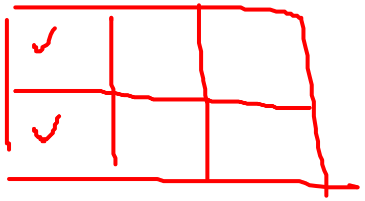
OR

```
void pass_1darray(int a[]) {
  cout << “Array at zero: ” << a[0] << endl;
}
```

Multidimensional Arrays

- data_type array_name[rows][cols];
  - int array[2][3];
  - int array[4][2][3];
  - int array[2][4][2][3];
- What are examples of these?
  - 2-D – Matrices, Spreadsheet, Minesweeper, Battleship, etc.
  - 3-D – Multiple Spreadsheets, (x, y, z) system
  - 4-D – (x, y, z, time) system

Initializing 2-D Arrays



stride

- **Declaration:** int array[2][3] ={{0,0,0},{0,0,0}};
- **Individual elements:**

```
array[0][0]=0;
array[0][1]=0;
array[0][2]=0;
array[1][0]=0;
array[1][1]=0;
array[1][2]=0;
```

- **Loop:**

```
for(i = 0; i < 2; i++)
  for(j = 0; j < 3; j++)
     array[i][j]=0;
```

- Why do we need multiple brackets?

Reading/Printing 2-D Arrays

- Reading Array Values

```
for(i = 0; i < 2; i++) {
  for(j = 0; j < 3; j++) {
    cout << “Enter a value for ” << i << “, ” << j << “: ”;
    cin >> array[i][j];
  }
}
```

- Printing Array Values

```
for(i = 0; i < 2; i++)
  for(j = 0; j < 3; j++)
     cout << “Array: ” << array[i][j] << endl;
```

Passing a 2-D Array (Static)

```
int main() {
    int array[5][5];
    …
    pass_2darray(array);
    OR
    pass_2darray(array, 5);
    …
}
void pass_2darray(int a[5][5]) {
    cout << “Array at zero: ” << a[0][0] << endl;
}
OR
void pass_2darray(int a[][5], int row) {
    cout << “Array at zero: ” << a[0][0] << endl;
}
```

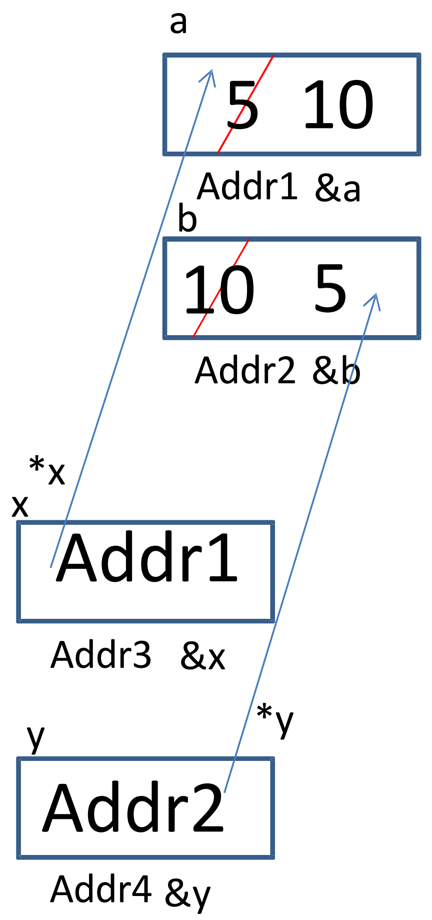
C/C++ Pointers

- Pointers == variables that hold memory addresses
- Variable declaration: `int a = 5;`
  - Creates a variable on the stack of size int with the value 5
- Pointer declaration: `int *b = &a;`
  - Creates a pointer variable on the stack which can hold an address of an int and sets the value of the pointer (the address the pointer points to) to the address of `a`
- Dereferencing Pointer: `cout << *b << endl;`
  - Dereference: access the value stored in the memory address held by a pointer
  - Will print the value stored at the address which `b` points to

- Every pointer points data of a specific data type

C++ Pointers

```
void swap(int *, int *);
int main() {
    int a = 5, b = 10;
    swap(&a, &b);
    cout << “a: ” << a << “b: ” << b;
}
void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}
```



Pointer and References Cheat Sheet

- &
  - If used **in a declaration** (which includes function parameters), it **creates and initializes** the reference.
    - Ex. void fun (int &p); //p will refer to an argument that is an int by implicitly using *p (dereference) for p
    - Ex. int &p=a; //p will refer to an int, a, by implicitly using *p for p
  - If used **outside a declaration**, it means **"address of"**
    - Ex. ptr=&a; //**fetches the address of** a (only used as rvalue!!!) and store the address in ptr. (ptr is a pointer variable)

Pointer and References Cheat Sheet

- *
    - If used **in a declaration** (which includes function parameters), it **creates** the pointer.
        - Ex. int *p; //p will hold an address to where an int is stored
    - If used **outside a declaration**, it **dereferences** the pointer
        - Ex. *p = 3; //**goes to the address** stored in p and stores a value
        - Ex. cout << *p; //**goes to the address** stored in p and fetches the value
- Check point: How to separate the following into two statements?

```
int *p = &a; //declare an int pointer and initialize it to &a
```