CS 162
Intro to Computer Science II

Lecture 2
Review (cont.)
1D & 2D static array
1/12/24
Odds and Ends

• Due Monday midnight:
  • Design exercise + Design doc
  • Quiz 1 (unlocked after today’s lecture)

• Lab 1: instead of 3 additional pts, you are allowed to make up for **full points** before your lab time next week
  • Get checked off by TAs during office hours
Lecture Topics:

• Finish Review
• 1D & 2D static array
C++ If/else and switch statements

```cpp
if (a == 0) {
    /* Do something. */
}
else if (b != 0) {
    /* Do something different. */
}
else {
    /* Do a third thing altogether. */
}

switch(grade) {
    case 'A' :
        cout << "Excellent!" << endl;
        break;
    case 'B' :
    case 'C' :
        cout << "Well done!" << endl;
        break;
    case 'D' :
        cout << "You passed!" << endl;
        break;
    case 'F' :
        cout << "Try again!" << endl;
        break;
    default :
        cout << "Invalid grade!" << endl;
}
```
C++ Loops

• C++:
  • for, while, do-while

```c++
int i;
for (i = 0; i < 32; i++) {
    /* Do something 32 times. */
}
```

```c++
while (i != 16) {
    /* Do something repeatedly until i is 16. */
}
```

```c++
doi{
    /* Do something repeatedly until i is 16. */
}while (i != 16);
```
C++ Functions

- Label:
  1. Function declaration/prototype
  2. Function call
  3. Function definition
  4. Function name
  5. Parameter(s)
  6. Argument(s)

```cpp
#include <iostream>

using namespace std;

float cal_avg(float num1, float num2);

int main()
{
    float total = 0;
    float count = 0;
    cout << "Enter total: ";
    cin >> total;
    cout << "Enter count: ";
    cin >> count;
    float average = cal_avg(total, count);
    cout << "Avg.: " << average << endl;
    return 0;
}

float cal_avg(float num1, float num2){
    return num1/num2;
}
```
C++ Pass by Value

```cpp
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;
}
```
C++ References

- Reference == a variable that refers to a particular memory address
- Reference declaration: `int i = 4; int &i_ref = i;`
  - A reference MUST be initialized
  - Once initialized, the memory address referred to by a reference variable can’t change
    - i.e. `i_ref` above must always refer to the address of `i`.
    - **Quick check**: what will the following code print?
      ```cpp
      int a = 7, b = 2, &ref = a;
      cout << a << " " << ref << endl; // prints
      ref = b;
      cout << a << " " << ref << endl; // prints
      ```
      -> Trying to make a new assignment to a reference changes its value
C++ Pass by Reference

```cpp
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;
}
```
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];
    • 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:

<table>
<thead>
<tr>
<th>Array index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>
Passing a 1-D Array to functions

```c
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;
}
```
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
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

- **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;
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

```cpp
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)
• *
  • 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