CS 161
Introduction to CS I
Lecture 29
The Virtual Edition

Tech Tips

- We will start at 2:00 p.m. (you can't hear me yet)
- Please mute your microphone.
  To ask a question, click "raise hand".
- **This meeting will be recorded.**

Hi! I'm still here ☺

Oregon State University
College of Engineering

Week 10 tips

- **Proficiency demo**: This has been converted to a regular lab, with a score of 0, 1, or 10
  - If you did not get a 10, your course grade will **not** be capped at 72% as originally indicated
  - There will be no in-person makeup proficiency exam (lab 10)
- Extra credit (for final exam): **survey of course materials (available on Canvas until midnight today)**

Oregon State University
College of Engineering

Week 10 tips: Final exam

- Final exam: Monday, 3/16, 6-7:50 p.m., on Canvas
  - All T/F and multiple choice (no short answer)
  - You have 1 hour and 50 minutes from when you start, up to 8:15 p.m. on 3/16
    (extra time in case you have a delay getting started; still only 1 hour 50 minutes for your exam)
  - Canvas will auto-submit your exam if you are still working on it at 8:15 p.m. on March 16
- I will have virtual office hours today via Zoom

Oregon State University
College of Engineering

Assignment 6 questions?

- Worth 80 points
  - Worth doing if any previous assignment earned < 80 points
  - Worth doing if you want practice with recursion ☺
  - Goal: extend the train_car struct (linked list) to allow passengers to board the train, then simulate a train journey

```
/* Structure defining a train car */
struct train_car {
  string kind; /* Engine, regular car (_***_), or Caboose */
  train_car* next_car; /* pointer to the next train car */
};
```

Final Exam Review Topics

- Data types and min/max ranges
- Expressions
- Operators
- Conditional statements
- Loops
- Random numbers
- Variable scope and shadowing
- Functions
- References
- Pointers
- 1-D arrays
- Dynamic memory allocation
- C-style strings
- 2-D arrays

Minimum and maximum values

Type	Minimum	Maximum
short (16 bits)	-32,768 -2^{b-1}	+32,767 $2^{b-1}-1$
unsigned short	0 0	65,535 $2^{b}-1$
int (32 bits)	-2,147,483,648	+2,147,483,647
unsigned int	0	4,294,967,295
long (64 bits)	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807
unsigned long	0	18,446,744,073,709,551,615
float	-3.4e38	3.4e38
double	-1.8e308	1.8e308

From Midterm 1

- Data types and min/max ranges
  - base types: `bool, char, short, int, long, float, double`
  - signed vs. unsigned
- Expressions
  - Parentheses: `12 / (3 + 1)`
  - Integer vs. floating point math:

    `(17-4) / 2` vs. `(17-4) / 2.0`

From Midterm 1

- Operators
  - Arithmetic: + - * / % ++ --
  - Relational: < <= > >= == !=
  - Logical: && || !
  - **Indexing: []**
  - **Memory: &(address-of) *(deref) .(member) ->(deref+member)**
  - Precedence https://en.cppreference.com/w/cpp/language/operator_precedence

Operator precedence

a++ a-- [] . ->
! ++a --a *p &a
* / %
+ -
< <= > >=
== !=
&&
||
= += -= *= /= %=

From Midterm 1

- Conditional statements
  - if-then
  - switch
  - break
- Loops
  - for
  - while
  - do-while
  - break
  - When to use each?

From Midterm 1

- Random numbers
  - Generate random numbers between 20 and 25 (inclusive)
  - Generate random numbers between -3 and 5 (inclusive)
- Variable scope (visibility) and shadowing

From Midterm 1

- Random numbers
  - Generate random numbers between 20 and 25 (inclusive)

```
rand()%6 + 20
```

  - Generate random numbers between -3 and 5 (inclusive)

```
rand()%9 - 3
```

- Variable scope (visibility) and shadowing

```
int m = 3;
if (m > 0) {
  int m = 43;
  cout << m++ << endl;
}
cout << m << endl;
```

Functions

- Function declaration vs. definition?
- Parts of a function declaration/definition?
- How to call a function?
- Pass by value vs. pass by reference

Functions

- Function declaration vs. definition?

  Declaration has return type, name, parameters; definition has code body

- Parts of a function declaration/definition?

  Return type, name, names and types of parameters

- How to call a function?

  retval = fn_name(argument1, argument2, …);

- Pass by value vs. pass by reference

  Value: make a copy; reference: pass the address (can modify value)

Functions

- What is function overloading?
- What is a case where function overloading is ambiguous?
- What are default arguments?
- Where must they appear in the function parameter list?

Functions

- What is function overloading?

  Same function name but different number or type of parameters
- What is a case where function overloading is ambiguous?

  Different return types but same parameter types
- What are default arguments?

  Placeholder values that will be used if the caller does not specify a value
- Where must they appear in the function parameter list?

  At the end of the parameter list

References and Pointers

- How do you declare a reference to another variable (char d)?
- How do you declare a pointer?
- How do you assign a pointer to point to an existing variable (d)?
- What are 2 ways to print the value in d?
- How do you print the value p points to?

References and Pointers

- How do you declare a reference to another variable (char d)?

```
char& z = d;
```

- How do you declare a pointer?

```
char* p = NULL;
```

- How do you assign a pointer to point to an existing variable (d)?

```
p = &d;
```

- What are 2 ways to print the value in d?

```
cout << d << endl;
```

```
cout << z << endl;
```

- How do you print the value p points to?

```
cout << *p << endl;
```

Oregon State University
College of Engineering

References versus Pointers

- Do not confuse "reference" (a data type) with "pass by reference" (something that happens when you call a function)
- Reference: an alias to some variable (permanent)
  - `int& r = s;`
  - Can assign new values to `r` (which is `s`), but cannot make `r` be an alias to another variable later
  - Must be initialized when declared
- Pointer: stores the address of some variable
  - `int* p = &s;`
  - Can change what address `p` contains (where it points to) anytime
  - Can be declared, then initialized later

1-dimensional arrays

- How do you declare a static array (e.g., of shorts)?
- How do you print item at index 3 in an array?
- If you print the name of the array (cout << arr), what is displayed?
- If you dereference the array (*arr), what do you get?
- How do you pass an array to a function?

1-dimensional arrays

- How do you declare a static array (e.g., of shorts)?

```
short array[4];
```

- How do you print item at index 3 in an array?

```
cout << array[3] << ednl;
```

- If you print the name of the array (cout << arr), what is displayed?

```
Memory location (address) of first item (array[0])
```

- If you dereference the array (*arr), what do you get?

```
Value of first item (array[0])
```

- How do you pass an array to a function?

```
fn(array);
```

Dynamic memory allocation

- What is the difference between the stack and the heap?
- When would you use the heap?
- How do you allocate memory (e.g., an integer) from the heap?
- How do you free the memory for an integer on the heap?

Dynamic memory allocation

- What is the difference between the stack and the heap?

  Stack is statically allocated (in advance); heap is dynamically allocated.

- When would you use the heap?

  To allow memory consumption to grow and shrink as needed; sizes (or numbers of items) are not known in advance.

- How do you allocate memory (e.g., an integer) from the heap?

```
int* d = new int;
```

- How do you free the memory for an integer on the heap?

```
delete d;
```

Dynamic memory allocation

- How do you allocate a 1-D array from the heap (e.g., short)?
- How do you free memory for a 1-D array on the heap?

Oregon State University
College of Engineering

Dynamic memory allocation

- How do you allocate a 1-D array from the heap (e.g., short)?

```
short* array = new short[17];
```

- How do you free memory for a 1-D array on the heap?

```
delete [] array;
```

C-style strings

- What kind of array is a C-style string?
- What library do you #include to access C-style string functions?

- What special item must a C-style string have? Why?

- `cin >> c_string;` reads user input and stops when?

- `cin.getline(c_string, 10);` reads how many characters from the user into `c_string`?

Oregon State University
College of Engineering

C-style strings

- What kind of array is a C-style string? `char[]`
- What library do you #include to access C-style string functions?
  `#include <cstring>`
- What special item must a C-style string have? Why?
  Null terminator ('\0' character), so functions know when string ends
- `cin >> c_string;` reads user input and stops when?
  Stops at first whitespace (space, tab, newline, etc.)
- `cin.getline(c_string, 10);` reads how many characters from the user into `c_string`?
  9 characters and adds the null terminator '\0' to make 10

2-dimensional arrays

- How do you declare a static 2-D array (e.g., 4x5 double)?

- This memory is laid out in row-major or column-major order?
- How do you allocate memory for a dynamic 2-D array?

- How do you free memory for a dynamic 2-D array?

Oregon State University
College of Engineering

2-dimensional arrays

- How do you declare a static 2-D array (e.g., 4x5 double)?

```
double array[4][5];
```

- This memory is laid out in Row-major or column-major order?
- How do you allocate memory for a dynamic 2-D array?

```
double** array = new double*[4];
for (int i=0; i<4; i++)
  array[i] = new double[5];
```

- How do you free memory for a dynamic 2-D array?

```
for (int i=0; i<4; i++)
  delete [] array[i];
delete [] array;
array = NULL;
```

2-dimensional arrays

- Given a 2-D (5x3) static array of ints, what type should be in the function definition to accept it?

- Given a 2-D (5x3) dynamic array of ints, what type should be in the function definition to accept it?

2-dimensional arrays

- Given a 2-D (5x3) static array of ints, what type should be in the function definition to accept it?

```
void my_fun(int arr[][3]);
void my_fun(int arr[5][3]);
```

- Given a 2-D (5x3) dynamic array of ints, what type should be in the function definition to accept it?

```
void my_fun(int** arr);
void my_fun(int* arr[]);
```

Oregon State University
College of Engineering

Structs and Recursion

- See practice questions on website

Week 10 (and the course) nearly done!

- ❑ Proficiency demo -> Lab 10
- ❑ Review and study for the **final exam**
- ❑ **Assignment 6** (due **Saturday, March 14**)

Hang in there – stay healthy and safe!