CS 161

Introduction to CS I

Lecture 27

- Command-line arguments
- File input and output

Oregon State University
College of Engineering

Week 10 tips

- Proficiency demo!
- Check Canvas for any missing grades
  - Notify cs161-020-ta@engr.orst.edu by Wednesday (3/11)
  - Your Canvas grade may not be your final course grade
- Final exam: Monday, 3/16, 6-7:50 p.m., LINC 128
  - All T/F and multiple choice (no short answer)
  - Review Midterm 1 and 2 solutions
  - See additional practice questions for structs and recursion (website)
  - No Thursday review session: review in class instead on Friday

Assignment 6: Train Journey

- 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

See lec27-stack-heap.cpp

A note about the stack vs. heap

- I want 1,000,000 train_cars. Where can I get them?

```
/* Static allocation */
train_car my_train[1000000];
```

```
/* Dynamic allocation */
train_car* my_train = new train_car[1000000];
delete [] my_train;
my_train = NULL;
```

See lec27-stack-heap.cpp

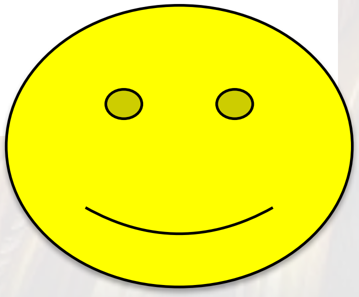
A note about the stack vs. heap

- I want 1,000,000 train_cars. Where can I get them?

```
/* Static allocation */
train_car my_train[1000000];
```

Seg fault

```
/* Dynamic allocation */
train_car* my_train = new train_car[1000000];
delete [] my_train;
my_train = NULL;
```



- => The heap is bigger than the stack

See lec27-recur-train.cpp

Oregon State University
College of Engineering

Give the user control over size of train

- Prompt user for n_cars

```
/* Create my train */
train_car* my_train = new train_car;
my_train->kind = "Engine";
my_train->next_car = NULL; /* be safe! */

cout << "How many cars to add to the train? ";
int n_cars;
cin >> n_cars;

add_cars(my_train, n_cars);
```

- Great for running test cases… unless you have to test many times

See lec27-recur-train-args.cpp

Oregon State University
College of Engineering

Give the user control over size of train

- Instead of waiting to type input each time, make it part of the command line
  - `./lec27-recur-train-args 1`
    `EngineCaboose`
  - `./lec27-recur-train-args 3`
    `Engine_***__***_Caboose`
  - `./lec27-recur-train-args 5`
    `Engine_***__***__***__***_Caboose`

See lec27-recur-train-args.cpp

Give the user control over size of train

- Instead of waiting to type input each time, make it part of the command line

Number of arguments

Array of char*, one per argument

```
int main(int argc, char* argv[]) {
  train_car* my_train = new train_car;
  my_train->kind = "Engine";
  my_train->next_car = NULL;
  int n_cars = atoi(argv[1]);
  add_cars(my_train, n_cars);
  ...
}
```

See lec27-recur-train-args.cpp

Oregon State University
College of Engineering

Give the user control over size of train

- argc: number of arguments
- argv: array of C-style strings
  - argv[0] = name of executable
  - argv[1] = first user-specified argument
  - …
- Convert C-style string to integer with atoi()
  - `int n_cars = atoi(argv[1]);`
- Likewise, atof() for floats

See lec27-recur-train-args.cpp

Good practice: check argc first

```
/* Expect and require argc == 2 (one user argument) */
if (argc != 2) {
    cout << "Usage: " << argv[0] << " n_cars" << endl;
    return 1; /* signal an error */
}
```

- To see the return value of the last command in linux:
  - **`echo $?`**

Oregon State University
College of Engineering

Your turn

- What is the value of **argc** if the user entered this command to run a program?

  ./my_prog the quick brown fox

- What does the 2-D array (**argv**) look like?

Working with files

- File = linear sequence of characters
- Stream = channel on which data is sent or received
  - `cin`: channel connected to keyboard
  - `cout`: channel connected to screen
- To work with files, create a file stream
  - `#include <fstream>`
  - `ifstream in_stream;`
  - `ofstream out_stream;`

See lec27-files.cpp

Write to an output file stream

- It works just like cout

```
ofstream out_stream;
out_stream.open("my_output.txt");
out_stream << "I am writing to a text file." << endl;
out_stream.close();
```

See lec27-files.cpp

Oregon State University College of Engineering

Read from an input file stream

- It works just like cin

```
string w;
int n_words = 0;
in_stream.open("my_output.txt");
while (in_stream >> w) {
    n_words++;
}
in_stream.close();
cout << "Read " << n_words << " words from file." << endl;
```

Using files with command-line arguments

- `./count_words input.txt`
- `./write_opera output.txt`
- `./translate input_english.txt output_piglatin.txt`

Oregon State University
College of Engineering

Minute paper

- What can you do now that you could not have done at the start of the term?
  - Not what do you know or have heard of
  - What **skill** or **ability** do you have?
  - Programming? Design? Testing? Debugging?

Oregon State University
College of Engineering

Week 10 begins!

- ❑ Demonstrate your proficiency in lab! Flex your muscles!
- ❑ Read:

  Args: https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/

  File I/O: http://www.doc.ic.ac.uk/~wjk/C++Intro/RobMillerL4.html
- ❑ Review and study for the **final exam**
- ❑ **Assignment 6** (due **Saturday, March 14**)

See you Wednesday!