# Coding Assignment 5

# Learning Outcomes of this Assignment

1. compiling/linking multi module programs using a makefile
2. reading command line arguments
3. throwing/catching exceptions
4. opening files and reading their contents into various C++ containers
5. using C functions in a C++ program
6. mutex
7. map
8. vector of object pointers
9. new
10. this
11. threads
12. static variables in objects
13. operator overloading
14. friend functions

# Notes

Before you can increment the static `ImDone` variable you will need to initialize it and reserve the space at the beginning of `TrickOrTreater.cpp`

```
int TrickOrTreater::ImDone = 0;
```

# Notes

This syntax

```
TrickOrTreater TOT(TOTname, Houses);
TrickOrTreaters.push_back(TOT);
```

is not equivalent to

```
TrickOrTreaters.push_back(TrickOrTreater{TOTname, Houses});
```

```
TrickOrTreater.h:18:7: note: ‘TrickOrTreater::TrickOrTreater(const
TrickOrTreater&)’ is implicitly deleted because the default
definition would be ill-formed:
```

# Notes

Remember that `strtok()` requires a character array. When you use `getline()` to read the file, you are getting a `string`.

You cannot give `strtok()` a `string`.

Look at using `c_str` to go from `string` to character array,

# Notes

You MUST set `argv[1]` to the Trick or Treater file.

You MUST set `argv[2]` to the House file.

You must set `argv[3]` to the Candy Rankings file.

The GTA will expect your program to accept the files in that order and if your program does not work with that order, then you will automatically receive a 0 since your program won't run.

# Notes

You will need to change the `House.h` file and the `TrickOrTreater.h` files to get your code to work. You need to change them as specified. You should not be adding extra code or changing the functionality.

The assignment will be graded by compiling your `cpp` files with the `.h` files that were used to created the templates that were provided with the assignment.

Your versions of `House.h` and `TrickOrTreater.h` will NOT be used for grading.


Terminal - student@maverick: /media/sf_VM/CSE1325/CA5
File Edit View Terminal Tabs Help
student@maverick:/media/sf_VM/CSE1325/CA5$ ./Code5_1000074079.e TOT.txt HOUSES.t
xt CANDY.txt


Terminal - student@maverick: /media/sf_VM/CSE1325/CA5
File Edit View Terminal Tabs Help
student@maverick:/media/sf_VM/CSE1325/CA5$ ./Code5_1000074079.e TOT.txt HOUSES.t
xt CANDY.txt

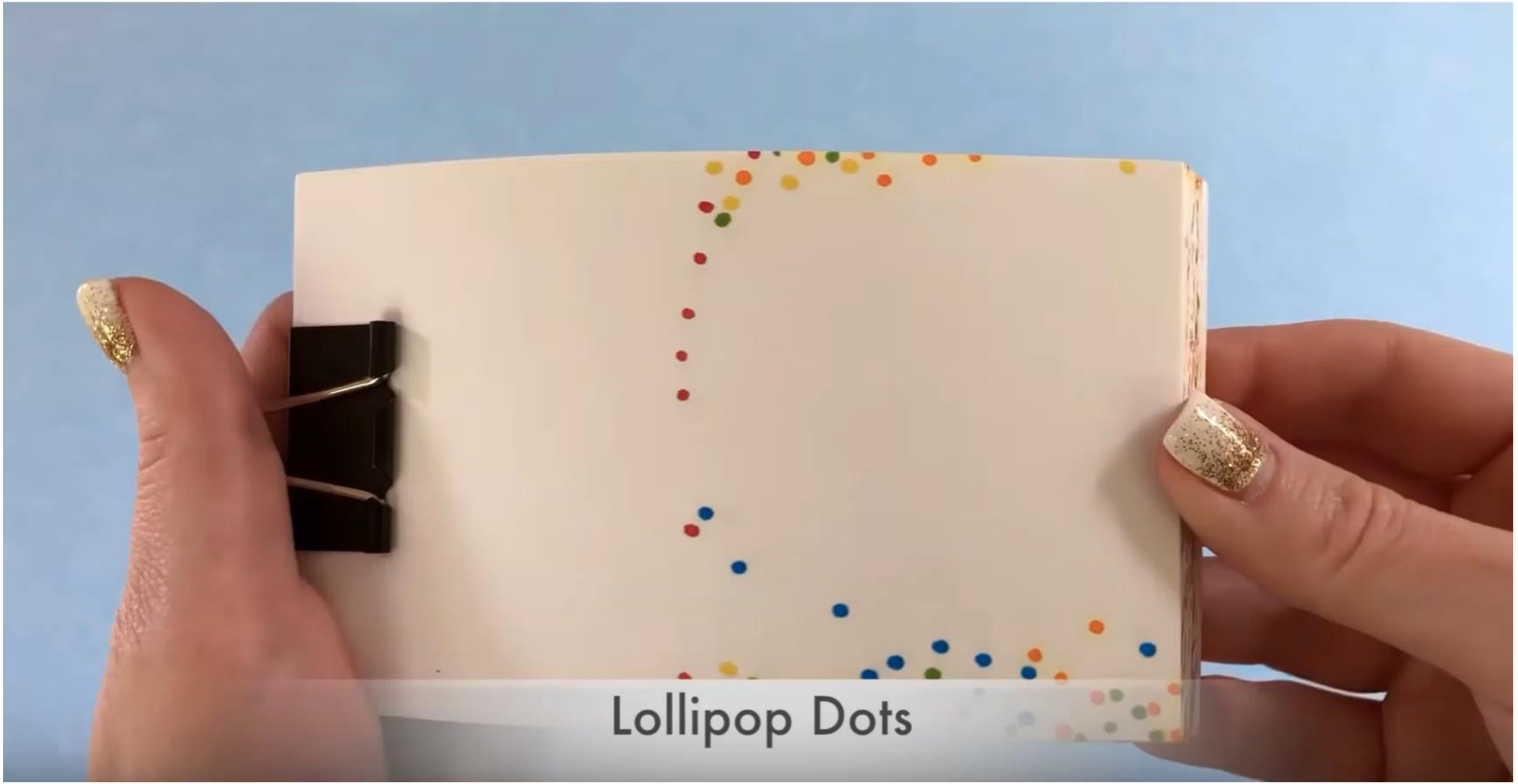
# Flip Book Effect

We are emulating movement on the screen by using the animation technique of a flip book.

A flip book or flick book is a booklet with a series of images that very gradually change from one page to the next, so that when the pages are viewed in quick succession, the images appear to animate by simulating motion or some other change.

https://en.wikipedia.org/wiki/Flip_book

Watch the following video to see several examples of movement/animation created using a flip book.


Lollipop Dots

If the output of the program is redirected to a file using >, then we can look at all of the output using an editor and by using more.

The file output.txt contains 503,130 lines which means the program printed 503,130 lines.

It takes a lot of printing to get the movement effect.

# Step 1

You will need to create 3 input files for your program. I have attached samples of all 3 files to the assignment. You should create your own files for testing. A different set of files will be for testing during the grading process.

A file of trick or treater's names - I suggest you use short names (10 characters per name) to make the display work well. Sample file is named TOT.txt.

A file of houses – this is just a list of names that will be displayed as the houses visited by the trick or treaters. Sample file is named HOUSES.txt. Names should be under 10 characters and the file should only contain at most 6 names; otherwise, the screen display will wrap.

A file of candy where each line is a candy name followed by a pipe symbol (|) and then a ranking for the candy. There can be as many candies listed in the file as you want – just make sure you have included all numbers from 1 to the number of lines in your file. Sample file is named CANDY.txt.

Remember that EOL conversions between Windows/Mac are different from UNIX and you may need to change them in your file. Notepad++ has the ability to change the EOL (Edit->EOL Conversion->Unix(LF). If you are not using Notepad++, then you can run your file through this conversion statement directly in your VM.

```
cat file.txt | tr '\r' '\n' | tr -s '\n' > file.translated.txt
```

This takes a file named `file.txt` and translates the EOL characters to LF and creates a new file called `file.translated.txt`.

```
 1 Charlie
 2 Lucy
 3 Linus
 4 PigPen
 5 Snoopy
 6 Patty
 7 Sally
 8 Shermy
 9 Rerun
10 Schroeder
11 Frieda
12 Marcie
13 Franklin
14 Violet
15 Woodstock
```

```
1 Frank
2 Mummy
3 Ghost
4 Ghoul
5 Goblin
6 Dracula
```

```
 1 KitKat|1
 2 M&Ms|2
 3 Snickers|3
 4 Milky Way|4
 5 Almond Joy|5
 6 Smarties|6
 7 AirHeads|7
 8 Skittles|8
 9 Jolly Rancher|9
10 Twix|10
11 Starburst|11
12 Butterfinger|12
13 Reese's|13
```

# Step 2

You will need 3 source files in your makefile. Your makefile will need to compile `Code5_xxxxxxxxxx.cpp` (referred to as just `Code5.cpp` after this point), the `TrickOrTreater.cpp` and the `House.cpp` and link them into one executable named `Code5.e`.

You will submit 4 files in a zip file named `Code5_xxxxxxxxxx.zip`. **As always, if your `makefile` does not compile your code or your code compiles with warnings/errors, then you will be assigned a grade of 0 regardless of the rest of your work. The GTA's will not alter your `makefile`/code to make the compile work.**

`Code4_xxxxxxxxxx.cpp`

`TrickOrTreater.cpp`

`House.cpp`

`makefile`

You will not submit the 3 data input files or either of the header files (`House.h` and `TrickOrTreater.h`) – the GTA's will grade with files that I will provide them. Do not make any assumptions about the contents of the files other than the restrictions listed above.

## Step 3

Copy your function to get command line arguments from the previous assignment into this new assignment. If the function does not detect 4 command line arguments, then it should throw an invalid argument exception. `main()` should catch that exception and exit the program if any of the command line arguments are missing. The function will read 3 file names from the command line (file of Trick or Treater names, house names and candy rankings). Function should store `argv[1]`, `argv[2]` and `argv[3]` in appropriate file names that are passed back to `main()`.

# Step 4

Create a map to hold the candy rankings. The ranking from the file should be the map's key and the candy's name should be the value of the map. Open the candy file and insert the information from the file into the Candy Ranking Map. You are required to use the C function `strtok()` to parse the lines from the candy file. Suggestion : temporarily add a range based for loop to print out the Candy Ranking Map to ensure that this code is functioning properly. Remove this temporary code once you prove your map is working.

```
 1 KitKat|1
 2 M&Ms|2
 3 Snickers|3
 4 Milky Way|4
 5 Almond Joy|5
 6 Smarties|6
 7 AirHeads|7
 8 Skittles|8
 9 Jolly Rancher|9
10 Twix|10
11 Starburst|11
12 Butterfinger|12
13 Reese's|13
```

| Key | Value |
|---|---|
| 1 | KitKat |
| 2 | M&Ms |
| 3 | Snickers |
| 4 | Milky Way |

# Step 5

Using the provided `House.h`, create `House.cpp`. A `House` object is constructed with a name and a Candy Ranking Map. A House has a member function called `ringDoorbell()` that does the following

Locks the `door` mutex

Adds an * to the passed in `ostringstream` (this is the * that shows up when a trick or treater rings a doorbell at a house).

Sleeps the thread for 3 seconds

Randomly picks a number between 1 and the number of candies in the Candy Ranking Map

Unlocks the `door` mutex

Returns the name of the candy associated with the random number in the Candy Ranking Map.

# Step 6

In `main()`, create a vector of pointers to `House`. While reading through the file of house names, use `new` to instantiate `House` objects. Each `House` object is constructed with its name and a copy of the Candy Ranking Map. This is also a good time to form the house names heading shown in the program's output. I took 11 – the length of the house's name and put that number of spaces between each name on the output screen to space them evenly. Suggestion : temporarily add a range based for loop to print out the vector of pointers to `House` objects to ensure that this code is functioning properly.

```
          Frank       Mummy       Ghost       Ghoul       Goblin      Dracula

..........*..........*..........*..........*..........*..........*Charlie
..........*..........*..........*..........*..........*..........*Lucy
```

So why do we need to use `new` and a `vector` of pointers to `House`?

Why not just construct `House` objects and add them to a vector of `Houses`?

The `mutex` class is a synchronization primitive that can be used to protect shared data from being simultaneously accessed by multiple threads.

`mutex` offers exclusive, non-recursive ownership semantics:

- A calling thread *owns* a `mutex` from the time that it successfully calls either `lock` or `try_lock` until it calls `unlock`.
- When a thread owns a `mutex`, all other threads will block (for calls to `lock`) or receive a `false` return value (for `try_lock`) if they attempt to claim ownership of the `mutex`.
- A calling thread must not own the `mutex` prior to calling `lock` or `try_lock`.

The behavior of a program is undefined if a `mutex` is destroyed while still owned by any threads, or a thread terminates while owning a `mutex`. The `mutex` class satisfies all requirements of *Mutex* and *StandardLayoutType*.

`std::mutex` is neither copyable nor movable.

What happens when an object is constructed and then added to a vector using `push_back`?

As we saw when we had the constructors and destructors print messages, `push_back` copies the object into the vector. The original object was then destroyed when we went out of scope of the construction.

Our `House` object contains a mutex (each `House` needs to lock/unlock its own door); therefore, we cannot instantiate objects and put them into a vector because that would require copying them.

Using `new` creates the object and gives back a pointer to the object which we can then safely add to a vector.

# Step 7

Using the provided `TrickOrTreater.h`, create `TrickOrTreater.cpp`. A `TrickOrTreater` object is constructed with a name and a list of `House` pointers. `TrickOrTreater.cpp` contains several member functions

`getName()` – returns the name of the trick or treater

`startThread()` - see Step 9 for more information

`joinThread()` – see Step 10 for more information

`GoTrickOrTreating()` – see Step 7A for more information

`Walk()` – see Step 7B for more information

`getPath()` – see Step 7C for more information

# Step 7A

Member function `GoTrickOrTreating()` is the function called when the thread is started.  It controls everything else the trick or treater does.

This function moves the trick or treater between houses, rings the doorbell at each house and increments the chosen candy in the trick or treater's candy bucket.  It also tracks which house the trick or treater is heading towards.

# Step 7A

Inside a range based for loop over the list of houses,

a `speed` is determined by picking a random number between 1 and 500.

`Walk()` is called and is passed `speed`

`ringDoorbell()` is called and is passed `path`. The return value of `ringDoorbell()` is the name of the candy randomly chosen and that name is used to key on that candy in `Bucket` and increment it.

`ringDoorbell()` returns the string `KitKat` to `GoTrickOrTreating()` where it is used with `Bucket`.

```
Bucket["KitKat"]++
```

## Step 7B

`Walk()` is used to create a different "walking" speed for each trick or treater as they travel between houses. Sometimes, they walk slowly and other times they run – their speed is based on random numbers.

Member function `Walk()` uses a for loop to stream single dots ‘.’ into `ostringstream path` one at a time.

Between outputting each dot, the thread sleeps for the `speed` value passed into `Walk()` from `GoTrickOrTreating()`.

See Step 7A for more information on how `speed` is calculated in `GoTrickOrTreating()`.

**Step 7C**

Data member `path` is an `ostringstream`; therefore, member function `getPath()` uses `path.str()` to return a string containing the trick or treater's current path.

As the tricker or treater "walks" (as the thread processes), their path is being added to at random intervals.

The path displayed on the screen is the current state of the object's path variable when `main()` grabs it via `getPath()` for printing to the screen.

# Step 8

In `main()`, create a vector of `TrickOrTreaters`. While reading through the file of trick or treater names, create a temporary trick or treater object and use `push_back` to add it to the vector. Each object is instantiated with a name and the list of houses.

```
vector<TrickOrTreater>TOTs;
```

```
 1 Charlie
 2 Lucy
 3 Linus
 4 PigPen
 5 Snoopy
 6 Patty
 7 Sally
 8 Shermy
 9 Rerun
10 Schroeder
11 Frieda
12 Marcie
13 Franklin
14 Violet
15 Woodstock
```

# Step 9

Using a range based for loop, call `startThread()` for all objects in the vector of `TrickOrTreaters`.

Each `TrickOrTreater` object contains a pointer to a thread called `TOTThreadPtr`. Function `startThread()` will instantiate a thread using `new` and store the pointer returned by `new` in `TOTThreadPtr`.

```
void TrickOrTreater::startThread()
{
    TOTThreadPtr = new std::thread(&TrickOrTreater::GoTrickOrTreating, this);
}
```

# Step 10

Use a range based for loop to call `TrickOrTreater`'s member function `joinThread()`. Use the `this` pointer to join each object's thread with `main()`.

```
void TrickOrTreater::joinThread()
{
    this->TOTThreadPtr->join();
}
```

# Step 11

In `main()`, while everyone is not done with trick or treating, print

enough newlines to clear/scroll the screen

the house list heading

use a range based for loop to print out the current version of each of the trick or treater's `path` and `name` (using `getPath()` and `getName()`)

sleep the `main()` thread for 5 milliseconds

In `main()`, while everyone is not done with trick or treating…

How do we tell if everyone is done trick or treating/has gotten candy from the last house on their list?

```
public :
    TrickOrTreater(std::string, std::vector<House*>);
    std::string getName();
    void startThread();
    void joinThread();
    void GoTrickOrTreating();
    void Walk(int);
    std::string getPath();
    static int ImDone;
```

After completing the loop over all `TrickOrTreater`'s, increment `ImDone` to show that this thread has visited all houses and is done trick or treating.

When we instantiate a class object, each object gets its own copy of all normal member variables.

Member variables of a class can be made `static` by using the `static` keyword.

Unlike normal member variables, `static` member variables are shared by all objects of the class.

Although you can access `static` members through objects of the class, `static` members exist even if no objects of the class have been instantiated.

Static members are created when the program starts and destroyed when the program ends.

It is better to think of `static` members as belonging to the class itself, not to the objects of the class.

Because `ImDone` exists independently of any class objects, it can be accessed directly using the class name and the scope resolution operator

```
TrickOrTreater::ImDone
```

Since `TrickOrTreater::ImDone` is automatically initialized to 0, we can have each `TrickOrTreater` object increment it when the last house is reached.

In `main()`, we can keep running our "flip book" animation until `ImDone` is equal to the number of trick or treaters (size of `TreatOrTreater` vector).

```
while (TrickOrTreater::ImDone != TOTs.size() )
{
     print 34 newlines
     loop over trick or treater vector
          print trick or treater's current path and name
     sleep for 5 milliseconds before refreshing screen
}
```

# Step 12

Use a range based for loop over the `TrickOrTreater`'s vector to print out the contents of each object's candy bucket. Overload `<<` such that you can use just `cout <<` iterator and print the bucket's contents. Copy your overloaded `<< friend` from the previous assignment and modify to print out the contents of `Bucket` which is a `map`.

`Bucket` is a `map` where the `map`'s key is the name of a candy and the `map`'s value is the number of that candy received by the trick or treater.

```
Charlie's - 1 AirHeads,1 Jolly
Rancher,1 Milky Way,1 Snickers,2
Starburst,
```

| Key | Value |
|---|---|
| AirHeads | 1 |
| Jolly Rancher | 1 |
| Milky Way | 1 |
| Snickers | 1 |
| Starburst | 2 |