Abstract Data Types

Lecture 05 Summary

- Abstract Data Types
- Structures in C

By the end of this lecture, you will be able to describe the main components of an abstract data type.

You will also be able to create and manipulate structures in C.

Recall from last class:

- One (good) approach:
  - Find *entities* which exhibit *state*
  - Analyze how the state of each entity changes
  - Create *variables* (or data structures) to hold the state of the entities
  - Create *code* that describes how to change the state of the entities

Exercise: design an application for the iPhone to look for a movie playing nearby.

What are the *entities*?

In Python...

```
movie = { "ID":"19478173",
          "Title":"The Curious Case of Benjamin Button",
          "Tomatometer":"0.72",
          "Release Date":{2008, 12, 25},
          "Show Times":[["Chinook","7:30pm"],
                        ["Chinook","10:00pm"],
                        ["Crowfoot","7:05pm"],
                        ["Crowfoot","9:20pm"]]
        }
```

- How would you add a new movie?
- How would you modify a movie?

Abstract Data Type (ADT)

- Attributes
  - A specification of a group of data
- Methods
  - A specification of how that data can be manipulated

Structs in C

- C does not allow us to associate specific functions with a set of data.
- It *does* allow us to formally group a set of data.

Example

```
struct location
{
    float longitude;
    float latitude;
};
```


Structure Definition vs. Instance

- Structure Definition
  - Creates a new type (like `long`, `int`, `char`)
- Instance
  - A chunk of memory created from the "recipe" of the structure definition
  - Can have multiple instances

Structure Definition vs. Instance

```
long x;

struct location my_loc;
struct location my_loc2;
struct location my_loc3;
```


Accessing Members

```
struct location my_loc;

my_loc.longitude = 51.08;
my_loc.latitude = 114.13;
```

Exercise (together): Write a function that creates a new "location" and a main program to test that function.

Example

```
struct address
{
    char street[50];
    char city[50];
    char province[50];
    char postal_code[8];
};

struct address *new_address()
{
    struct address *temp =
        malloc(sizeof(struct address));

    temp->street[0] = '\0';
    temp->city[0] = '\0';
    temp->province[0] = '\0';
    temp->postal_code[0] = '\0';

    return temp;
}
```

```
int main(int argc, char **argv)
{
    if (argc < 5)
    {
        printf("Usage: %s ...\n",
               argv[0]);
        exit(1);
    }

    struct address *my_address =
        new_address();

    strcpy(temp->street, argv[1]);
    strcpy(temp->city, argv[2]);
    strcpy(temp->province, argv[3]);
    strcpy(temp->postal_code, argv[4]);

    printf(
 "Address is:\n\t%s\n\t%s, %s\n\t%s\n",
        my_address->street,
        my_address->city,
        my_address->province,
        my_address->postal_code);
}
```


Example

```
> ./address.exe "123 Easy Street" "Calgary" "Alberta" "T1T 2T3"
```

- How much space is needed?
- How much space is allocated?

How can we fix this problem?

Possible Solution

```
struct address
{
    char *street;
    char *city;
    char *province;
    char *postal_code;
};
```


Rewrite

```
struct address *new_address()
{




}
```


Write

```
void setStreet(struct address *add, char *value)
{




}
```


Adding Functionality

```
float distance(struct location *first,
               struct location *second)
{




}
```

What are the two major components of an abstract data type?

Lecture 05 Summary

- Abstract Data Types
  - Attributes
  - Methods
- Structures in C
  - Allow grouping of variables

Next Class

- Pointers + ADTs
- Dynamic Memory Allocation
- Linked Lists