CSc 352
Function Pointers

Benjamin Dicken

Function Pointers

- Can call functions by their pointer (address), rather than by name
- Can store pointers to functions in variables!

Two Functions

```
int summation(int a) {
  int result = 0;
  while(a > 0) {
    result = result + a;
    a--;
  }
  return result;
}

int factorial(int a) {
  int result = 1;
  while(a > 0) {
    result = result * a;
    a--;
  }
  return result;
}
```

Same return value, same number of parameters and parameter types

Function Pointers

```
int main(int argc, char** argv) {
  int result = 0;
  int (*action)(int);
  int value = 5;

  action = factorial;
  if (argc >= 2 && argv[1][0] == 's') {
    action = summation;
  }

  result = (*action)(value);
  printf("%d\n", result);

  return 0;
}
```

This is a function pointer variable

Assigning a function pointer variable

Call function via pointer

Activity

Function Pointers

- Why are function pointers useful?
- Think of particular examples where it could come in handy

Function Pointers for Behavior Customization

One use case of function pointers is to use it as a mechanism to customize the behavior of some task in your program

Example: Customizing how strings are matched in a search

```c
int main() {

  char test[] = "Trees trees TREES t r e e s tReEs";

  int r1 = number_of_matches(test, "trees", case_sensitive);
  printf("  case_sensitive: %d\n", r1);  // 1

  int r2 = number_of_matches(test, "trees", case_insensitive);
  printf("case_insensitive: %d\n", r2);  // 4

  int r3 = number_of_matches(test, "trees", case_sensitive_ignore_spaces);
  printf("case_sensitive_ignore_spaces: %d\n", r3);  // 2

  return 0;
}
```

Pass a function pointer into the function to customize what counts as a match

Function pointer as one of the parameters, customize the way matching happens

```
int number_of_matches(char* to_search_through,
                      char* search_term,
                      int(*compare)(char*, char*)) {
  int count = 0;
  for (int i = 0; to_search_through[i] != '\0'; i++) {
    count += compare(&to_search_through[i], search_term);
  }
  return count;
}
```

In this one, do a basic string equality check

```
int case_sensitive(char* base, char* term) {
  int i = 0;
  while (base[i] != '\0' && term[i] != '\0') {
    if (base[i] != term[i]) return 0;
    i++;
  }
  if (term[i] != '\0') return 0;
  return 1;
}
```

```c
int case_insensitive(char* base, char* term) {
  int i = 0;
  while (base[i] != '\0' && term[i] != '\0') {
    char b = base[i];
    char t = term[i];
    if (b >= 'A' && b <= 'Z') {
      b = b+32;
    }
    if (t >= 'A' && t <= 'Z') {
      t = t+32;
    }
    if (b != t) return 0;
    i++;
  }
  if (term[i] != '\0') return 0;
  return 1;
}
```

Add code to account for case-insensitivity

Ignore spaces, except for the first character

```c
int case_sensitive_ignore_spaces(char* base, char* term) {
  int i = 0;
  int j = 0;
  while (base[i] != '\0' && term[j] != '\0') {
    while (i != 0 && base[i] == ' ') i++;
    while (j != 0 && term[j] == ' ') j++;
    if (base[i] != term[j]) return 0;
    i++;
    j++;
  }
  if (term[j] != '\0') return 0;
  return 1;
}
```

Bonus: Can C be Object Oriented?

Object-Oriented

- In OO languages such as Python and Java, we can
  - Create classes, that define both **variables** and **functions**
  - Instantiates instances of classes (objects)
  - Call functions via an object with dot syntax

    ```
    Car x = new Car();
    x.honkHorn();
    ```
  - Can avoid directly accessing the variables, use getters and setters, etc

Object-Oriented

```java
public class Car {
  private String color;
  private String horn_sound;
  private int horse_power;
  private double latitude;
  private double longitude;

  public Car() { . . . }

  public void honkHorn() { . . . }
  public void showColor() { . . . }
  public void move (double lat, double  lng) { . . . }
}
```

```java
public static void main(String[] args) {
  Car x = new Car();
  x.honkHorn();
}
```

Object-Oriented

Can something similar be done in C?

Object-Oriented

Can something similar be done in C?

Yes, but it is much uglier

Define a Struct with Functions

```c
typedef struct Car {
  char* color;
  char* horn_sound;
  int horse_power;
  double latitude;
  double longitude;
  void (*honk_horn)(struct Car* this);
  void (*show_color)(struct Car* this);
  void (*move)(struct Car* this, double lat, double  lng);
} Car;
```

"member variables"

"methods"

Define the functions

```c
void honk_horn(struct Car* this) {
  if (this->horn_sound == NULL) {
    printf("honk!\n");
  } else {
    printf("%s\n", this->horn_sound);
  }
}

void show_color(struct Car* this) {
  if (this->color == NULL) {
    printf("red\n");
  } else {
    printf("%s\n", this->color);
  }
}

void move(struct Car* this, double lat, double lng) {
  this->latitude  = lat;
  this->longitude = lng;
}
```

Define a “constructor” with CPP

```
#define NEW_CAR(hp, lat, lng) \
 (Car) {           \
  NULL, NULL,      \
  hp, lat, lng,    \
  honk_horn,       \
  show_color,      \
  move             \
 };
```

Use it!

```c
int main(int argc, char** argv) {

  Car plain = NEW_CAR(200, 0, 0);

  plain.show_color(&plain);
  plain.honk_horn(&plain);
  plain.move(&plain, 1.0, 2.0);

  return 0;
}
```