Estruturas de Dados / Programação 2
Ponteiros para Funções

Márcio Ribeiro
marcio@ic.ufal.br
twitter.com/marciomribeiro

Now we need to compute the clients miles

Different rules to compute miles/points

- Suppose that there are similar steps for all airlines…
- … and different steps depending on each airline

```
int compute_miles(int flight_number,
                  int day,
                  int month,
                  int airline_code)
{
  // Similar steps for all airlines…

  switch (airline_code) {
    case 0:   //american airlines
    …
    case 27:  //srilankan airlines
  }
}
```

Let's create a function for each airline

- We can call them within the case statements…

```
int american_airlines(int flight_number) { … }
int air_berlin(int flight_number) { … }
int british_airways(int flight_number) { … }
…
int s7_airlines(int flight_number() { … }
int srilankan_airlines(int flight_number) { … }
```

- But how can we avoid the switch statement and make our code cleaner and better to read and understand?!

Can we call *compute_miles*…

- … and at the same time specify for which airline we need to do such a computation?!
- Instead of passing airlinecode, what about passing the entire function?!

```
int compute_miles(int flight_number,
                  int day,
                  int month,
                  airline function)
{
  // Similar steps for all airlines…

  call airline function
}
```

Pointers to Functions!

Pointer to a function

- When a function is compiled, we have an entry point
- When we call the function, the entry point is executed
- A pointer can contain the address of this entry point
- So, we can use the pointer to call the function

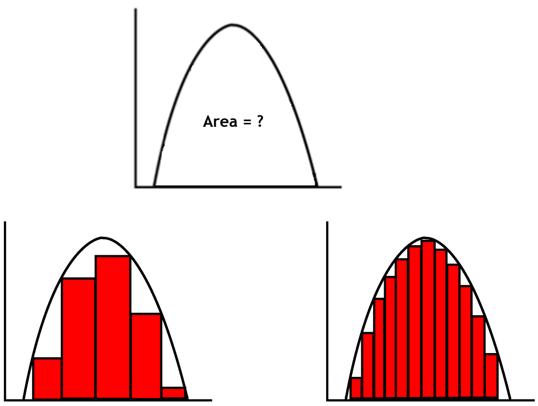
```
int american_airlines(int flight_number) { … }
int (*airline_function)(int);
airline_function = american_airlines;
```

Now…

- *compute_miles* has a function as parameter and we can easily call this function!

```
int compute_miles(int flight_number,
                  int day,
                  int month,
                  int (*airline_function)(int flight_number))
{
  // Similar steps for all airlines…
  int airline_result = (*airline_function)(flight_number);
}
```

Another example: areas




```
double integral(double (*f)(double x), double a, double b)
{
  double sum, dt;
  int i;

  sum = 0.0;
  dt = (b - a) / 100.0;
  for (i = 0;  i < 100;  i++)
    sum += (*f)(i * dt + a) * dt;

  return sum;
}

double square(double x)
{
  return x * x;
}

double cube(double x)
{
  return x * x * x;
}

printf("Integral = %f\n", integral(square, 2, 3));
printf("Integral = %f\n", integral(cube, 2, 3));
```

We call this Higher-order Function!

Higher-order function

- Takes one or more functions as input
- Common in functional programming (e.g., Haskell)

```
sum :: Int -> Int -> Int
sum x y = x + y

calc :: (Int -> Int -> Int) -> Int -> Int -> Int
calc f a b = f a b
```

Exercises

Exercise 1: map

- Takes two inputs (function and array) and then applies the function to every element of the array. This new array is returned
- Implement a function map
- Call the map function with the following functions:
  - Square
  - Factorial
- Now, call map passing an array and the square function; then, do the same for the factorial function

Exercise 1: solution in Haskell

```
myMap :: (Int -> Int) -> [Int] -> [Int]

myMap f [] = []

myMap f (a:as) = [f a] ++ myMap f as
```

Exercise 2: filter

- Takes two inputs (function and array), where function is a test. Filter chucks out any elements of the array that do not satisfy the test
- Implement a function filter
- Call the filter function with the following functions:
  - even
  - odd
- Now, call filter passing an array and the even function; then, do the same for the odd function

Exercise 2: solution in Haskell

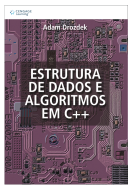
```haskell
myFilter :: (Int -> Bool) -> [Int] -> [Int]

myFilter f [] = []

myFilter f (a:as) =
   if (f a) then
      [a] ++ myFilter f as
   else
      myFilter f as
```

Why pointers to functions?
Why Haskell?

Recursive functions!
We're gonna need them
in many data structures!

References

Chapter 5



Chapter 1