Data types

Single pieces of information
- one integer – int
- one symbol – char
- one truth value – bool

Multiple pieces of information
- group of symbols – string
- group of anything – array
- group of multiple things – struct, class

Introducing: classes

- A class defines a new data type
- Each instance of a class is an object
- Each object can contain
  - Actions to perform (functions)
  - Information about the object (variables)

Class syntax – declaration

class Dog
{
  public:
    void Bark();
    void Eat(float foodQuantity);
    void Walk(float distance);
    float size, weight, location;
};

Class syntax – function definitions

void Dog::Bark()
{
  cout << "Woof woof!\n";
}

void Dog::Eat(float foodQuantity)
{
  weight+=foodQuantity/2;
  size+=foodQuantity/10;
}

Class dog

Example information
- Size
- Weight
- Location

Example actions
- Eat
- Walk
- Bark
Class syntax – create and use an object

```cpp
class Dog {
    int weight;
    int size;

    void Eat(int amount) {
        weight -= amount;
        size += amount;
    }

    void Bark() {
        // Bark function implementation
    }
}
```

```cpp
int main()
{
    Dog fido;
    fido.weight=40.5;
    fido.size=10;
    fido.Eat(20);
    cout << fido.weight << " " << fido.size << endl;
    return 0;
}
```

Typical program layout

```cpp
class Dog {
    . . .
};

int main()
{
    Dog fluffy;
    fluffy.Bark();
    . . .
}
```

```cpp
void Dog::Bark()
{
    . . .
}
```

The Dot Operator .
- Used for functions and data of individual objects
- `fido.Bark()`

The Scope Resolution Operator ::
- Used for functions of a class
- `Dog::Bark()`

Note: a function inside a class is called a “member function”

Multiple instances of a class

```cpp
int main()
{
    Dog rufus, spot;
    rufus.weight=40.5; rufus.size=10;
    spot.weight=30; spot.size=7.5;
    rufus.Eat(20); spot.Eat(2);
    cout << "New location for Rufus: 
      " << rufus.location << endl;
    // Will output location 8.4
    return 0;
}
```

Time to walk the dog...

```cpp
int main()
{
    Dog rufus;
    rufus.weight=35; rufus.size=7.2;
    rufus.location=5;
    rufus.Walk(3.4);
    cout << "New location for Rufus: 
      " << rufus.location << endl;
    // Will output location 8.4
    return 0;
}
```

public vs. private

- public: any function can see and use
- private: only visible to member functions

- Good style:
  - make all member variables private
  - use public functions to access and mutate variables
Class declaration, take 2

```cpp
class Cat {
public:
    void set(float inWeight, float inSize,
             float inLoc);
    float getSize();
    float getWeight();
    float getLocation();
    . . .
    void Walk(float distance);
private:
    float size, weight, location;
};
```

Function definitions – take 2

```cpp
void Cat::set(float inWeight,
              float inSize,
              float inLoc)
{
    weight=inWeight;
    size=inSize;
    location=inLoc;
}
```

Class usage – take 2

```cpp
int main()
{
    Cat feline1;
    feline1.set(5.5,20.1,2);
    cout << feline1.location; // Error
    cout << feline1.getLocation() << endl;
    return 0;
}
```

Bank account

Variables
- Name
- Current balance
- History of cash in (and out)

Functions
- Deposit:
  - Add entry to history
  - Update balance

```cpp
class Account {
public:
    void open(string inName);  
    void deposit(float money);  
    float getBalance();

private:
    string name;  
    float balance;  
    float history[1000];  
    int num_transactions;
};
```

```cpp
void Account::open(string inName){
    name=inName;
    balance=0;
    num_transactions=0;
}

void Account::deposit(float money) {
    if(money>=0) {
        history[num_transactions]=money;
        num_transactions++;
        balance = balance+money;
    } else {
        cout << "Error! " << "Negative deposit!\n";
    }
}
```
What does this do?

```cpp
int main()
{
    Account acc1;
    acc1.open("Tina");
    cout << acc1.getBalance() << endl;
    acc1.deposit(250);
    acc1.deposit(20.25);
    cout << acc1.getBalance() << endl;
}
```

Withdrawal function?

- How can we write `withdraw` function to reduce the money in our account?
- How can we prevent over-drawing?

Abstraction

- Function – a set of actions called by one word
- Class – a set of data held in one word

Information hiding

- So long as action/data unit acts correctly, we don’t need to know the details
- Hiding details can prevent accidents in programming (e.g., overdrewn account)

Constructor functions

- Can declare and initialize object simultaneously
  ```cpp
  int main() {
      Account acc1("Tina", 200.20);
  }
  ```
- Constructor function(s) defined to initialize object

Constructor definition

- Constructor has same name as class
- Constructor has no return type

```cpp
Account::Account(string inName, float inDollars) {
    name=inName;
    if(inDollars>=0) {
        balance=inDollars;
        num_transactions=0;
    } else {
        cout << "Error, negative dollar amount!";
        exit(1);
    }
}
```

Alternate constructor

Variable initialization can begin before `{ }` of constructor function as `variable_name(variable_value)`

```cpp
Account::Account() :
    name("John Doe"), balance(0),
    num_transactions(0)
{
    // Left empty
}
```