**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**

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

---

**Class syntax – function definitions**

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

void Dog::Eat(float foodQuantity)
{
    weight+=foodQuantity/2;
    size+=foodQuantity/10;
}
```
Class syntax – create and use an object

```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();
    ...
}
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 fido, spot;
    fido.weight=40.5; fido.size=10;
    spot.weight=30; spot.size=7.5;
    fido.Eat(20); fido.Eat(2);
    cout << fido.weight << " "
         << spot.weight << endl;
    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;
}
```

Exercises

- Write the `Walk` function
- Modify the `Walk` function so the dog loses 0.2 pounds for every foot he walks
- Let’s say dogs hide a bone at each location where they have stopped. Add an array `hiddenBones` that records each location where the dog object has hidden bones and modify `Walk` again to leave a record in `hiddenBones`
Class syntax – declaration

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

Class syntax – create and use an object

```cpp
int main()
{
  Dog fido;
  fido.weight=40.5;
  fido.Bark();
  // Set fido's location to 3
  ??? // output location
  // Have fido change location
  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.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;
};
```

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

void Account::deposit(float money) {
    if (money >= 0) {
        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?

Account review

Member variables
- name
- balance
- history,
  num_transactions

Member functions
- open
- deposit
- getBalance
- withdraw
- printHistory

New accessor function
- string getName()
Declaring/initializing

• We can declare and then initialize a variable
  
  ```
  int a;
  a=1;
  Account acc1;
  acc1.open("Tiana");
  ```

• Or we can declare and initialize together
  
  ```
  int b=1;
  Account ...?
  ```

Constructor functions

• Declaring and initializing object simultaneously
  
  ```
  int main() {
    . . .
    Account acc1("Tiana",200.20);
    . . .}
  ```

• Constructor function(s) defined to initialize object
  
  ```
  Account acc1="Tiana",200.20;
  ```

Constructor definition

• Constructor has same name as class
• Constructor has no return type
  
  ```
  Account::Account(string inName,
  float inDollars)
  {
    name=inName;
    balance=inDollars;
    history[0]=balance;
    num_transactions=1;
  }
  ```

Testing equality

• Simple variables can use ==
  
  ```
  int a=5, b=7;
  if(a==b)
    cout << "Variables equal\n";
  ```

• Complex variables cannot use ==
  
  ```
  Account acc1("Tiana",20.50),
  acc2("Jim",9.95);
  if(acc1==acc2) // Does not work!
    cout << "Accounts equal\n";
  ```

Testing object equality

• Can use functions to test equality
  
  ```
  Account acc1("Tiana",20.50),
  acc2("Jim",9.95);
  if(acc1.equals(acc2))
    cout << "Accounts equal\n";
  ```
Equality function

```cpp
bool Account::equals(Account accA)
{
    // test balance match
    if(balance!=accA.getBalance())
        return false;
    // test name match
    // don't worry about matching
    // histories
}
```

Testing array equality:

```cpp
int a[4]={4,-5,0,2}, b[4]={4,-5,0,2};
if(a==b) // Will not work correctly
    cout << "Equal arrays\n";

• Can use functions to test array equality
bool equalArrays(int arr1[], int size1,
                 int arr2[], int size2
{
}
```

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., overdrawn account)