Classes
TA: You Lu
Using user created classes

To declare a variable and initialize it we used a syntax like:

```c
int my_number = 17;
```

For classes we use a slightly different syntax.

To initialize an instance of a class we use a constructor.

Syntax:
```
class_name object_name(initial values);
```
Using objects

Objects usually have member functions. We have already seen how to use member functions with the string class.

The syntax is:

object_name.function_name(parameters if any);

You should be able to use a class in your program if one is given to you with a description of its member functions.
Classes are awesome!

• Classes make programming easier
• Classes shorten code
• Classes are reusable
• Classes reduce bugs
Declaring a class

Anatomy of a class declaration:

```cpp
class class_name{
    public:
    constructor declarations
    member function declarations

    private:
    member variables
};
```

Public section of the class. These are the constructors and member functions we can access via the dot operator.

Internal data of the class. These variables and functions are not accessible outside the class functions.
What goes into public section

Constructor declarations

Constructor is a function and needs to be declared. We need to have a prototype for the constructor.

Game g(15, 18, 100); This command evokes the constructor.

Member functions

Any member functions that we can call with the . operator are declared in the public section. Their prototypes are put into the public section.
What goes into the private section

Private variables

These are variables that store essential data of the class. The variables are only accessible by the class itself. We cannot access them (in main) via the dot operator.

Helper functions (private functions)

Sometimes a class will use internal functions that are not member functions themselves.
Game class example
Encapsulation

What is the idea behind this convoluted way of doing things?

One of the ideas is that we don't have to know how the variables in the class are stored or how the class manipulates the variables.

We don't know. We don't have to know. We don't care. This is the idea of encapsulation.

The programmer is isolated from the data and all manipulation done via function calls.
class DayOfYear
{
    public:
        void set(int newmonth, int newday);

    Private:
        int month;
        int day;
};

Note: This is just a bare bones example. Let's define the set function.

void DayOfYear::set(int newmonth, int newday)
{
    month=newmonth;
    day=newday;
}
int main()
{
  DayOfYear today;  // Declares a blank DayOfYear object

  today.set(2,15);  // initializes today by using the
                   // set member functions.
}

Ok this is all well and good. But I could have written:

today.set(435435,3249834);
void DayOfYear::set(int newmonth, int newday)
{
    if (newmonth >= 1 && newmonth <= 12))
        month=newmonth;
    else
    {
        cout << "Illegal month!";
        exit(1); // Ends program requires <cstdlib>
    }

    if (newday >= 1 && newday <= 31)
        day=newday;
    else
    {
        cout << "Illegal day";
        exit(1);
    }
}
Constructors

Constructor is a function that declares the instance of the class. It allocates the memory for the object and it may initialize the private variables.

There are two types of constructors. The default constructor which just creates the object and either leaves variables uninitialized or initializes always to default values.
Default constructor syntax:

Declaration:

class class_name
{
    public:
        class_name(); // Default constructor
...
};

Definition:

class_name::class_name()
{
    // Private variable initializations
}
Example

class DayOfYear{

public:
    DayOfYear(); // constructor

    /*
     * Other member functions
     */

private:
    int month;
    int day;
};
Default constructor definition

```cpp
DayOfYear::DayOfYear()
{
    month = 1;
    day = 1;
}
```
Example

How to use a default constructor:

```c
int main()
{

    DayOfYear today;

    ...

}

today is a DayOfYear object that our constructor creates and initializes month to be 1 and day to be 1.
```
Constructor syntax:

Declaration:

class class_name
{
    public:
        class_name(type par1, type par2...);
...

};

Definition:

class_name::class_name(type par1, type par2)
{
    // Private variable initializations
}
Constructors with arguments example

**Declaration:**
class DayOfYear  
{
  public:
      DayOfYear(int new_month, int new_day);
  
    /*
       Other member functions
    */

  private:
      int month;
      int day;
  
};

**Definition:**
DayOfYear::DayOfYear(int new_month, int new_day)
{
  month = new_month;
  day = new_day;
}
Using a constructor with arguments

```c
int main()
{
    DayOfYear today(5,3);
}
```
If you don't write any constructors the computer creates a default constructor for you!

This default constructor creates the object but does not initialize the variables.

You can have several constructors for your class. One constructor could be the default constructor and the second one that takes arguments.

If you write a constructor that takes arguments you no longer have the computer created default constructor.
#include <iostream>
using namespace std;

class Point
{
public:
    Point();
    Point(double new_x, double new_y);
    void move(double dx, double dy);
    double get_x() const;
    double get_y() const;

private:
    double x;
    double y;
};

Important concepts:
Public, Private
Member functions,
Variables
Accessors, Mutators
Constructors
Syntax
Constructor definitions

Point::Point()
{
    x=0.0;
    y=0.0;
}

Point::Point(double new_x, double new_y)
{
    x=new_x;
    y=new_y;
}

- This is a default constructor
- Constructors have the same name as the class.
- There is no return type
- Private variables are “global” to the class

- This is a constructor with parameters
- Parameters cannot have same name as private variables.
- We can have multiple constructors with same names!

How does computer know which one to use?
void Point::move(double dx, double dy)
{
    x+=dx;
    y+=dy;
}

double Point::get_x() const
{
    return x;
}

double Point::get_y() const
{
    return y;
}
```cpp
int main()
{
    Point P1;
    Point P2(1.0, 3.0);

    cout << "x coordinate of P1 is: " << P1.get_x() << ". The y coordinate of P1 is: " << P1.get_y() << "\n";

    cout << "x coordinate of P1 is: " << P2.get_x() << ". The y coordinate of P2 is: " << P2.get_y() << "\n";

    cout << "Moving P2..." << "\n";           

    P2.move(3.0, 2.0);

    cout << "x coordinate of P1 is: " << P2.get_x() << ". The y coordinate of P2 is: " << P2.get_y() << "\n";

    return 0;
}
```
When declaring member functions as const you have to make sure that your member function does not change the private variables!

This means that your const member function cannot call non-const member functions.

Even if the function you are calling from inside a const function does not change the private variables, the computer thinks that there is a possibility that the function you are calling changes the private variables. This would mean that your const function might be indirectly changing the private variables!
Recall

If you don't write any constructors the computer creates a default constructor for you! This constructor does not take any arguments.

You can create an instance of your class:
Point P;

This default constructor creates the object but does not initialize the variables.

You can have several constructors for your class. One constructor could be the default constructor and the second one that takes arguments.

If you write a constructor that takes arguments you no longer have the computer created default constructor. So you should write a default constructor yourself.
#include <iostream>
#include <string>
using namespace std;

class Product
{
public:
    Product();
    void read();
    bool is_better_than(Product b) const;
    void print() const;
private:
    string name;
    double price;
    int score;
};
Product::Product()
{
    name = "";
    price = 1;
    score = 0;
}
```cpp
void Product::read()
{
    cout << "Please enter the model name: ";
    getline(cin, name);
    cout << "Please enter the price: ";
    cin >> price;
    cout << "Please enter the score: ";
    cin >> score;
    string remainder; // Read remainder of line
    getline(cin, remainder);
}
```
bool Product::is_better_than(Product b) const
{
    if (price == 0) return true;
    if (b.price == 0) return false;
    return score / price > b.score / b.price;
}

• We compute score/price to see which product gives you better “bang for the buck”.
• We return true if the current product has better ratio than product b.
• Be sure to avoid division by zero!
• To call this function for Products a and b, use something like:

        if ( a.is_better_than(b) ) cout<<“Buy a!”;

• We can access the price of a because we are in the instance of a, however it is slightly strange that we can access the price of b just by writing b.price. Recall that non-member functions can’t get to price directly. We might expect that we would have to do b.get_price(). This is not necessary in this special case because b is also of type Product.
void Product::print() const
{
    cout << name
        << " Price: " << price
        << " Score: " << score;
}
```cpp
int main()
{
    Product best;
    bool more = true;
    while (more)
    {
        Product next;
        next.read();
        if (next.is_better_than(best))
            best = next;
        cout << "More data? (y/n) ";
        string answer;
        getline(cin, answer);
        if (answer != "y")
            more = false;
    }
    cout << "The best value is ";
    best.print();

    return 0;
}
```
What our program files look like now

Structure of the program looks like this:

• #include <libraries>
• using namespace std;
• class declarations { };
• class member functions
• function declarations
• program functions
• main routine
Having multiple constructors with same name is an example of something called function overloading.

You are allowed to have functions with the same names provided that:

1) They have different number of arguments

```java
int foo(int a, int b);
int foo(int a);
```

2) The types of arguments are different.

```java
int foo(double a);
int foo(int a);
```

It is not enough for return types to be different:

```java
void foo();
int foo();    // Illegal
```
```cpp
double average(double x1, double x2)
{
    return (x1+x2)/2;
}

double average(double x1, double x2, double x3)
{
    return (x1+x2+x3)/3;
}

Now from main I can call either function by:

    x = average(a,b);

or

    x = average(a,b,c);
Operators such as + - % == are nothing but functions that are used with different syntax from normal functions.

We don't write +(a,b) we write a+b.

Just as you can overload regular functions, you can overload operators.

Next time we will talk about how to define operators for classes.
Wouldn't it be nice if we could compare classes like we compare variables?

For example we wrote earlier for products a and b:

```java
if (a.is_better_than(b))
{
    // Do stuff
}
```

Instead we would like to write:

```java
if (a > b)
{
    // Do stuff
}
```
We need to define for the computer what it means to compare two classes. We do this by overloading the > operator.

In our class declaration we make a small change:

```cpp
class Product
{
public:
  Product();
  void read();
  // bool is_better_than(Product b) const;
  bool operator > (Product b) const;
  void print() const;
private:
  string name;
  double price;
  int score;
};
```
This is how we define operator member function

```cpp
bool Product::operator>(Product b) const
{
    if (price == 0) return true;
    if (b.price == 0) return false;
    return score / price > b.score / b.price;
}
```
while (more)
{
    Product next;
    next.read();
    // if (next.is_better_than(best))
    if (next > best)
        best = next;

    cout << "More data? (y/n) ";
    string answer;
    getline(cin, answer);
    if (answer != "y")
        more = false;
}
We can overload other operators as well. Without knowing we have already been using one other overloaded class operator:

```c++
    string name = "Homer " + "Simpson";
```

+ is overloaded for the string class.

We could overload `>=`, `==`, `*`, `+`, `-` even `%`. 
Passing classes to functions

When an instance of a class is passed to a function by default it is passed by value just as any ordinary variable.

If you want your functions to change your object you need to pass it by reference.

```cpp
void foo (Point &P)
{
P.move(1.0, 3.9);
}
```

There are other reasons to pass your object by reference. Making a copy of a class costs a lot of overhead.