Friend Function & Class

Friend Functions/Classes

`friends` allow functions/classes access to private data of other classes.

Friend Functions/Classes

Friend functions

A 'friend' function has access to all 'private' members of the class for which it is a 'friend'.

To declare a 'friend' function, include its prototype within the class, preceding it with the C++ keyword 'friend'.

Sample application that can benefit from friend classes/functions

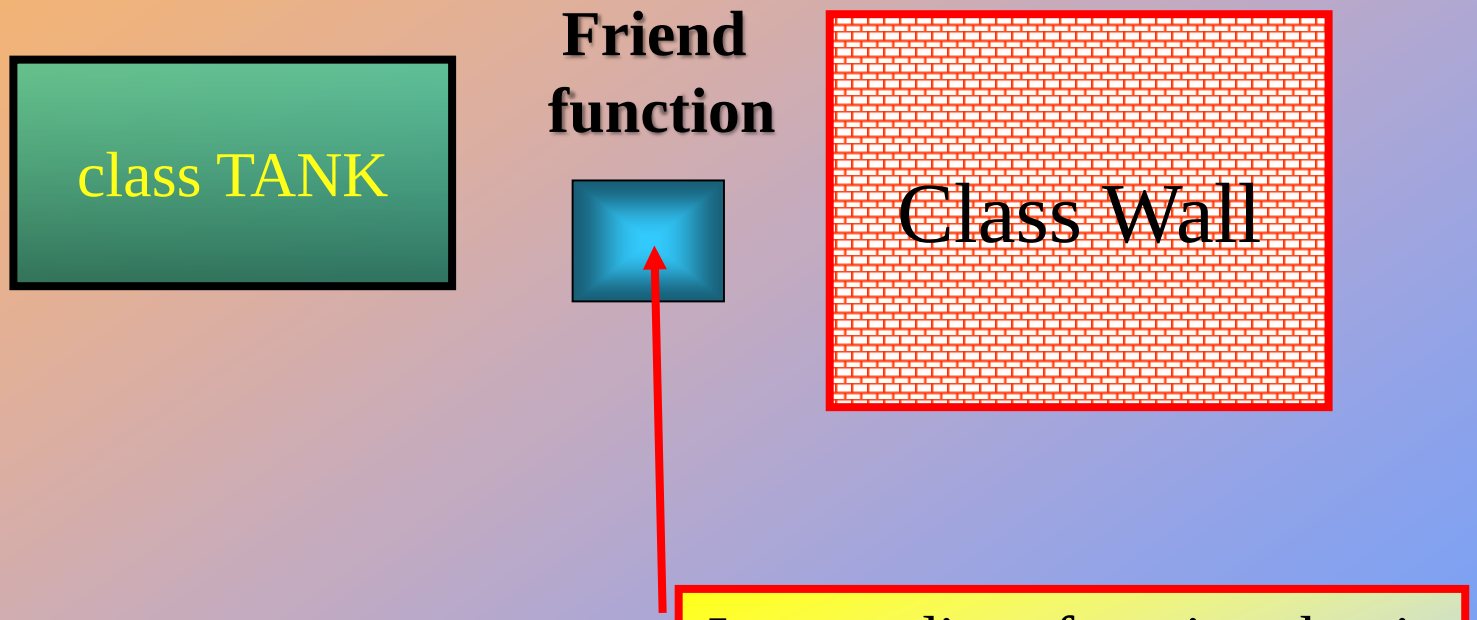
COLLISION PROBLEM

APPROACHES: SIMPLE RECTANGLE , SHRUNKEN RECT, SPRITE IMAGE

- One class for each moving object
- One instance of a class doesn't know the boundaries of other moving objects
- Data members of each object is hidden and protected from other moving objects

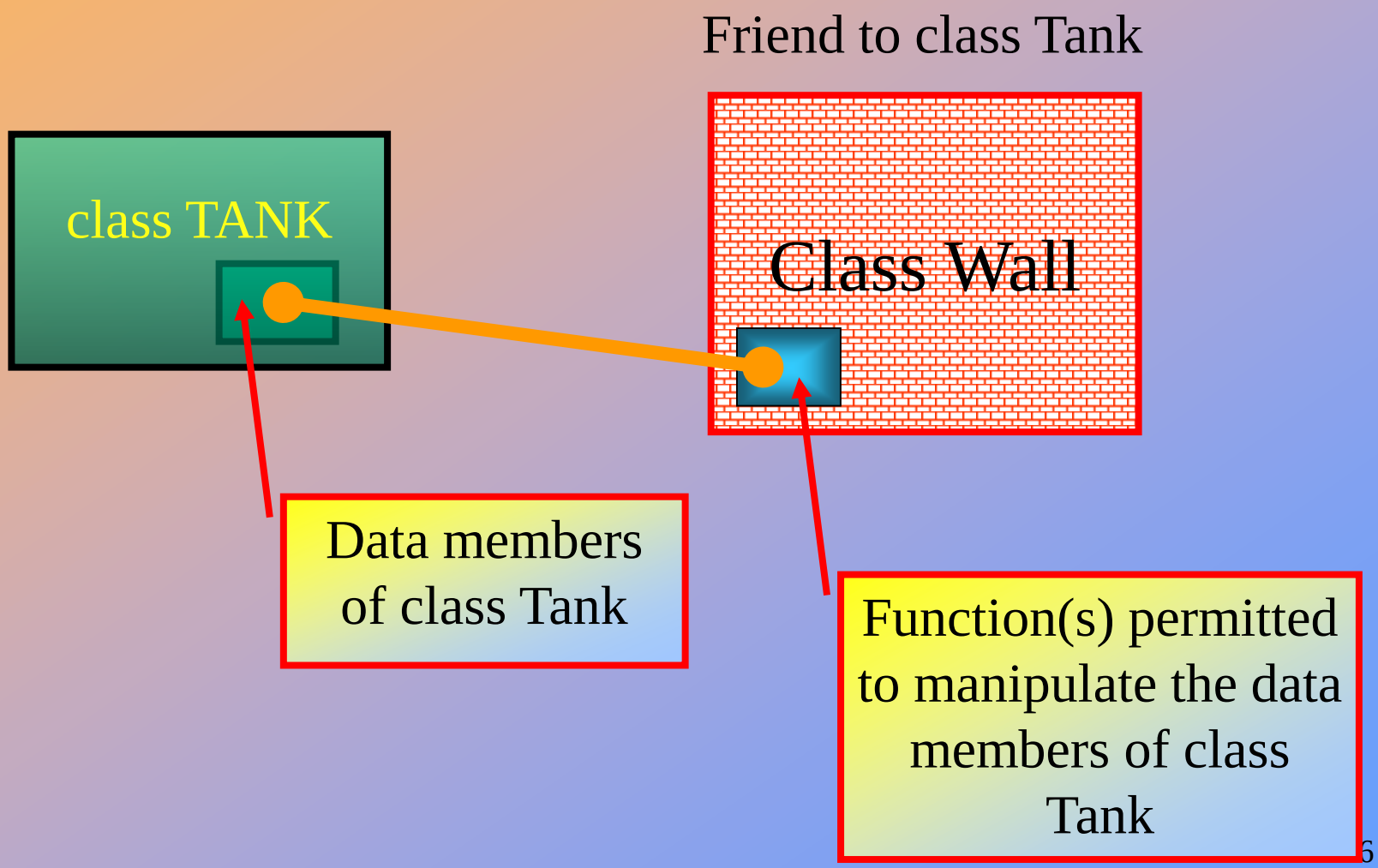
COLLISION PROBLEM

Friend functions/class by-pass object data hiding



COLLISION PROBLEM

Friend functions/class by-pass object data hiding

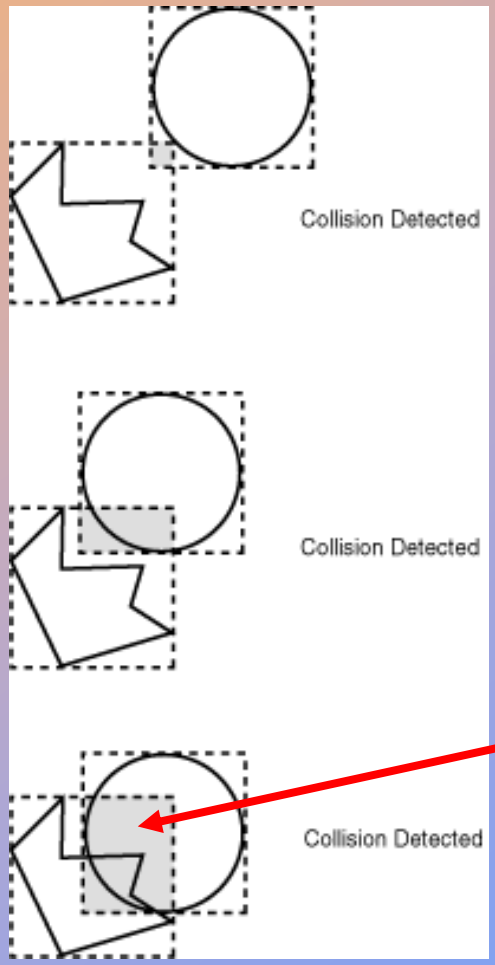


COLLISION PROBLEM

SIMPLE RECTANGLE APPROACH

How to detect collision between objects?

- **not convincing enough!**

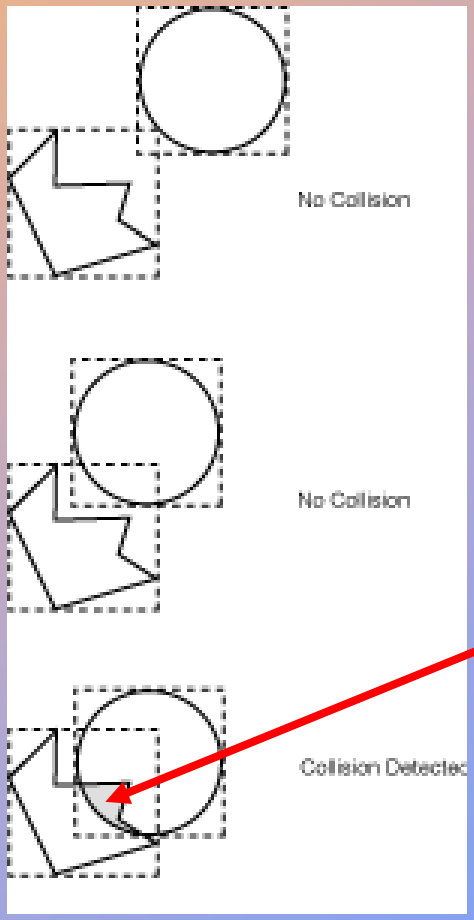


Overlapping region

COLLISION PROBLEM

SPRITE IMAGE APPROACH

How to detect collision between objects?



Overlapping region

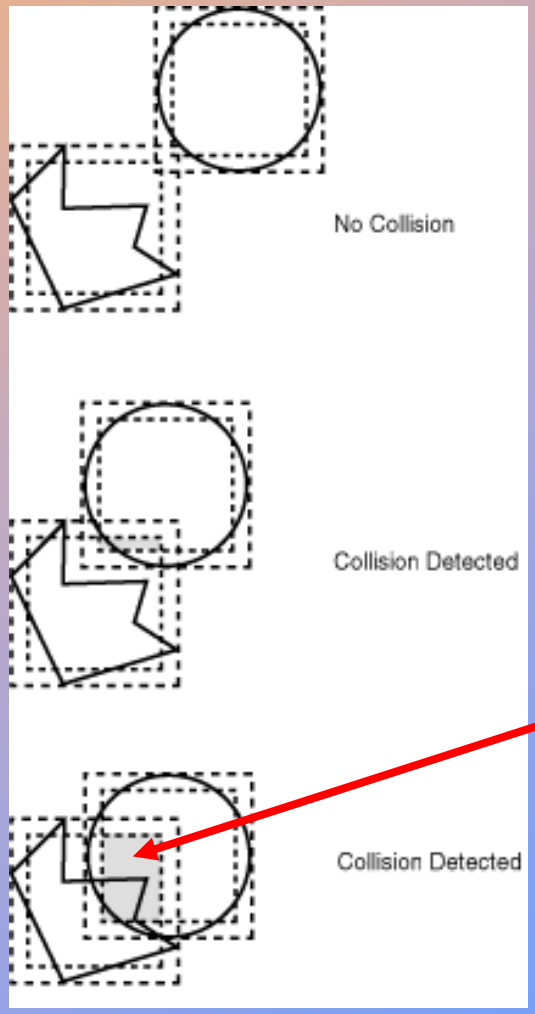
- **can be a major bottleneck in performance to get a decent animation**

COLLISION PROBLEM

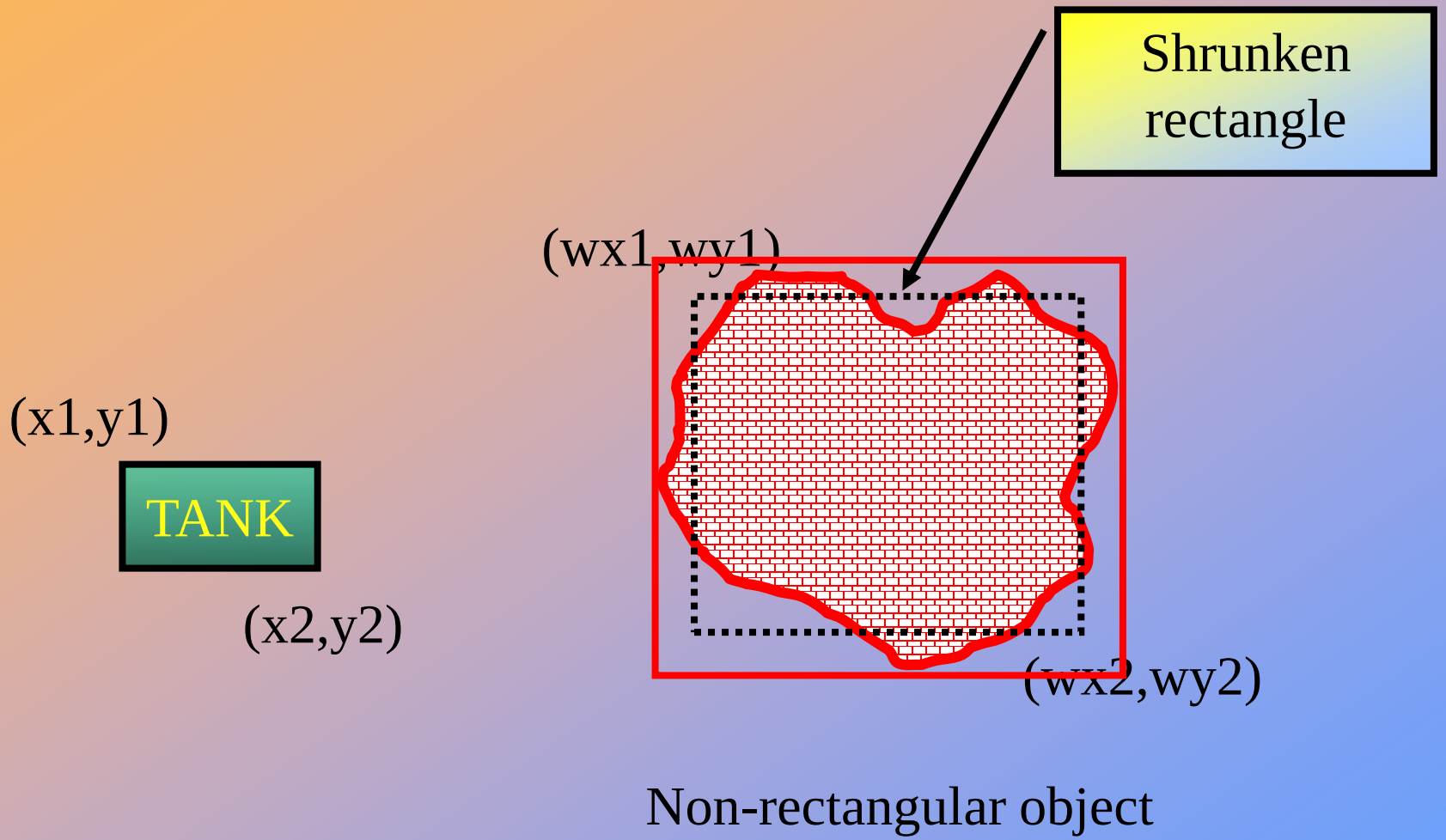
SHRUNKEN RECTANGLE APPROACH

How to detect collision between objects?

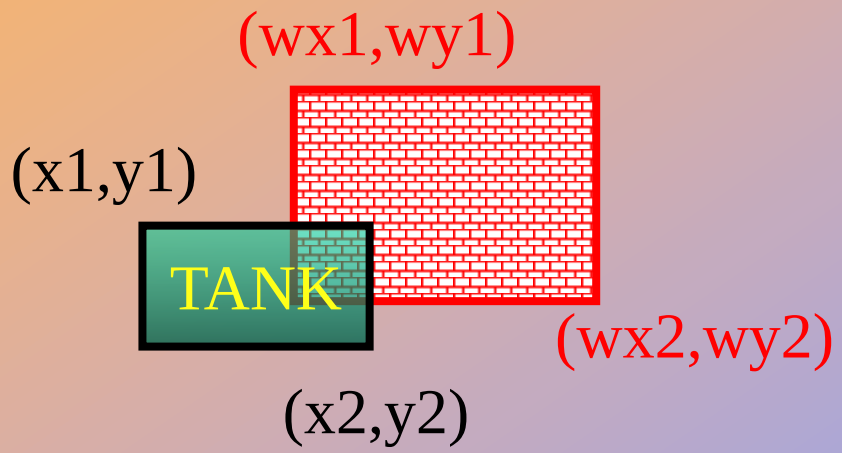
- **Fast**
- **Simple to implement**
- **perfect for a game with many moving objects**



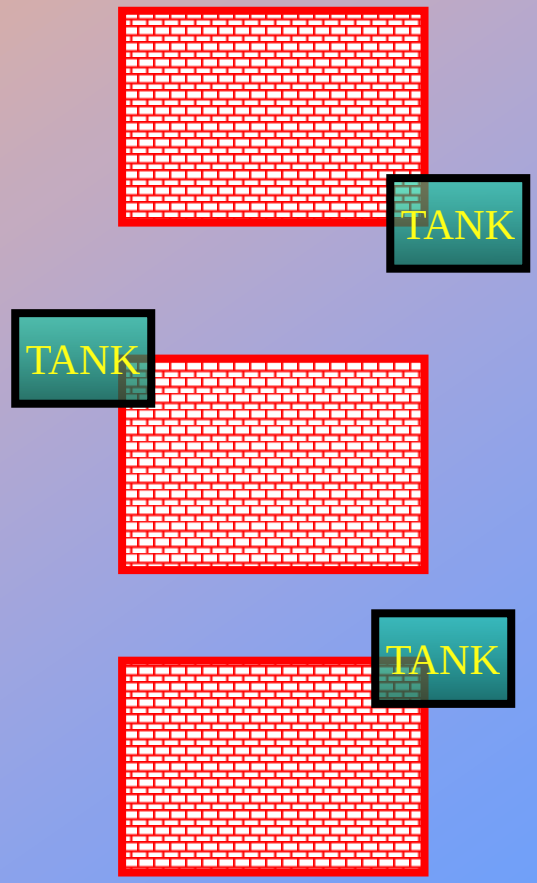
COLLISION PROBLEM



COLLISION PROBLEM



Sample Collision Cases:

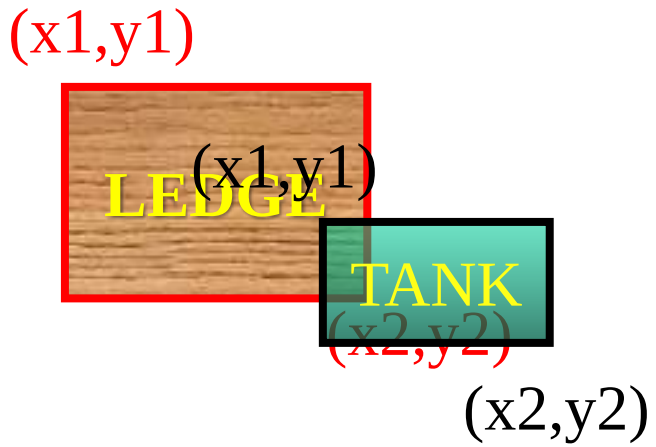


COLLISION PROBLEM

Sample Collision Cases:

(t.x1 >= ledge.x1) && (t.x1 <= ledge.x2)

Note: This example is only using the device system of coordinates and is therefore not scalable.

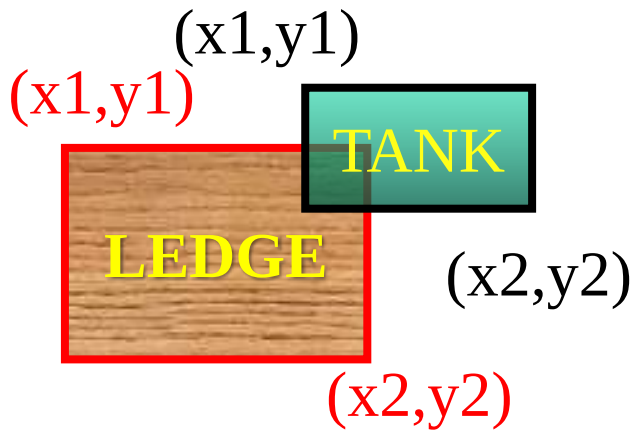



```
if(  ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2))    ||
     ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2))    )
```

COLLISION PROBLEM

Sample Collision Cases:

(t.x1 >= ledge.x1) && (t.x1 <= ledge.x2)



Note: This example is only using the device system of coordinates and is therefore not scalable.

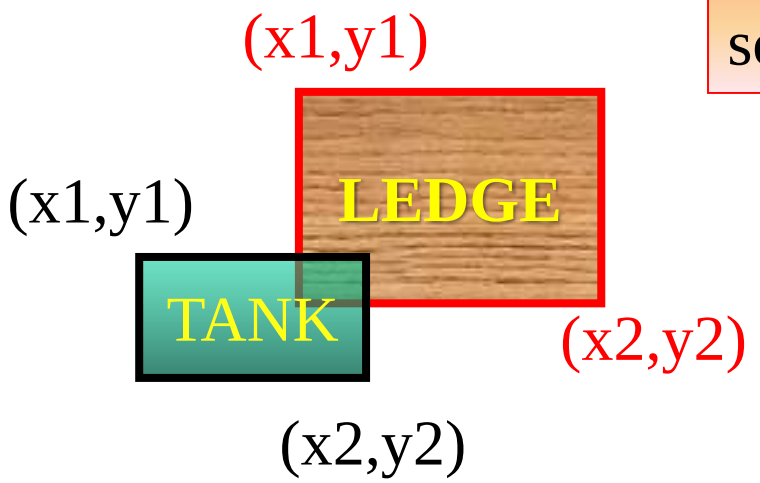
```
if(  ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2))    ||
     ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2))     )
```

COLLISION PROBLEM

Sample Collision Cases:

(t.x2 >= ledge.x1) && (t.x2 <= ledge.x2)

Note: This example is only using the device system of coordinates and is therefore not scalable.

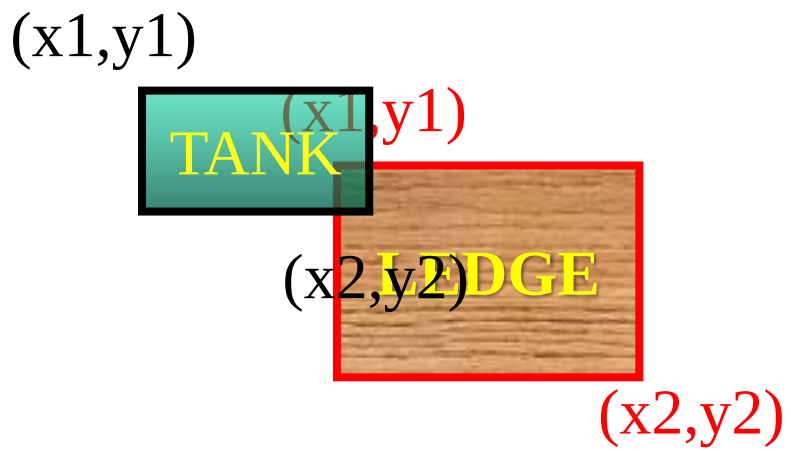



```
if(  ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2))    ||
     ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2))     )
```

COLLISION PROBLEM

Sample Collision Cases:

(t.x2 >= ledge.x1) && (t.x2 <= ledge.x2)



Note: This example is only using the device system of coordinates and is therefore not scalable.

```
if(  ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2))    ||
     ((t.x1 >= ledge.x1) && (t.x1 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y1 >= ledge.y1) && (t.y1 <= ledge.y2)) ||
     ((t.x2 >= ledge.x1) && (t.x2 <= ledge.x2) && (t.y2 >= ledge.y1) && (t.y2 <= ledge.y2))     )
```

Friend Functions/Classes

```
class T {
 public:
   friend void a();
   int m();
 private:  // ...
};

void a() {// can access
          // private data in T...}


class S {
 public:
   friend int T::m();
   //...
};

class X {
public:
   friend class T;
   //...
};
```

- Global function **a()** can access private data in **T**
- **m()** can access private data in **S**
- all functions of **T** can access private data in **X**

friends should be used with caution: they by-pass C++'s data hiding principle.

It is the responsibility of the code for which access is to be given to say who it's friends are - i.e. who does it trust!

Friend Functions/Classes

```
class Demo {
   friend void Change( Demo obj );
 public:
    Demo(double x0=0.0, int y0=0.0):x(x0),y(y0){}
    void print();
 private:
    double x; int y;
};

void Demo::print(){
   cout << endl << "This is x " << x << endl;
   cout << "This is y " << y << endl;
}

void Change( Demo obj ) {
   obj.x += 100;
   obj.y += 200;
   cout << "This is obj.x" << obj.x << endl;
   cout << "This is obj.y" << obj.y << endl;
}
```

Friend Functions/Classes

```
#include <iostream>
using namespace std;

const int DaysInMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
enum Months { unused, January, February, March, April, May, June,
              July, August, September, October, November, December };
const char *MonthNames[]={"Unused", "January", "February", "March", "April", "May",
        "June",
       "July", "August", "September", "October", "November", "December" };

class Date{
 friend bool before( Date  & d1, Date & d2 );

public:
 Date( int d=1, int m=1, int y=1970 ){  // do not use initialiser list in constructor
  setMonth( m);                    // as here we want to reuse the checks
  setDay(d);                       // in the set methods
  setYear(y);   // remember  month is  used to verify day validity
 }
 ~Date(){} // no special needs for destructor

 Date( const Date & date ){  // supply an explicit copy constructor
  day = date.day;
  month = date.month;
  year = date.year;
 }

 int getDay(){ return day; }
 void setDay( int d ){
  if( d > 0 && d <= DaysInMonth[month] ){
    day = d;
  }
  else{
    cerr << "Invalid  Day value " << d << endl;
    exit(1);
  }
 }

int getMonth(){ return month; }
 void setMonth( int m ){
  if( m >= 1 && m <= 12 ){
    month = m;
  }
  else{
    cerr << "Invalid  Month  value " << m << endl;
    exit(1);
  }
 }

 int getYear(){return year;}
 void setYear( int y ){
  // no restrictions placed on year
  year = y;
 }

 void print(){
  cout << day << " of " << MonthNames[month] << ", " << year << endl;
 }

private:
 int day;
 int month;
 int year;
};
```

- **Continued on next slide…**

Friend Functions/Classes

```
bool before( Date  & d1, Date & d2 );  // prototype

int main(){  // test the Calendar collection
  Date today;
  Date lecture11( 6, 8, 2008 );
  today.setMonth( August );
  today.setDay( 7 );
  today.setYear( 2008 );
  today.print();
  lecture11.print();
  if( before( lecture11 , today ) )
     cout << "lecture11 was before today" << endl;
  Date tomorrow( 8, 8, 2008 );
  if( before( tomorrow , today ) )
     cout << "tomorrow was before today" << endl;
  else
     cout << "tomorrow was not before today" << endl;
  return 0;
}

// return  true if   Date1 is before Date2
bool before( Date  & d1, Date & d2 ){
  if( d1.year < d2.year )
      return true;
  else if( d1.year == d2.year ){
      if( d1.month < d2.month )
         return true;
      else if( d1.month == d2.month ){
        if( d1.day < d2.day )
           return true;
      }
  }
  return false;
}
```

- We wanted the global before() function to have access to the internals of the Date class
- Date declares it as a friend function

- Example output:

4 of August, 2022

3 of August, 2022

lecture11 was before today

tomorrow was not before today

Friend Functions/Classes

Thank You.

Friend Functions/Classes

Questions

1. **Is there any difference between `List x;` and `List x();`?**
2. **Is the default constructor for `Fred` always `Fred::Fred()`?**
3. **Should constructors use "initializer lists" or "assignment"?**
4. **How do you know when to declare a function as a member function or a `friend` function in your class?**

Friend Functions/Classes

The following questions pertain to a class called Circle.

a) The only element in a circle considered unique is the radius. Write one line of code that declares the necessary data member.

b) In which section of the class does this data member get put?

c) Write the prototype for the constructor which takes a radius as its parameter.

d) In which section of the class does this constructor get put?

e) Write the function **definition** for the member function Area which computes the area of a Circle object. *(Just use 3.14159 for Pi. And remember that area of a circle of radius R is given by:* πR^2*.)*