Part 1: Testing

Testing in C++

We'll use **Catch2** for testing:

`https://github.com/catchorg/Catch2/blob/v2.x/docs/tutorial.md`

`https://tinyurl.com/catchv2header`

Testing in C++

We'll use **Catch2** for testing:

`https://github.com/catchorg/Catch2/blob/v2.x/docs/tutorial.md`

`https://tinyurl.com/catchv2header`

Save as `catch.h` or `catch.hpp`

Adding third-party source into your Git repo is usually not a great option, but it's reasonable in this case

Needs at least `--std=c++14`

Writing Tests

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

Writing Tests

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

Makes `TEST_CASE` and `CHECK` available

Writing Tests

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

main.cpp

```
#define CATCH_CONFIG_RUNNER
#include "catch.h"

int main(int argc, char **argv) {
  Catch::Session().run(1, argv);
  return 0;
}
```

Writing Tests

Must be before `#include "catch.h"`

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

main.cpp

```
#define CATCH_CONFIG_RUNNER
#include "catch.h"

int main(int argc, char **argv) {
  Catch::Session().run(1, argv);
  return 0;
}
```

Writing Tests

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

main.cpp

```
#define CATCH_CONFIG_RUNNER
#include "catch.h"

int main(int argc, char **argv) {
  Catch::Session().run(1, argv);
  return 0;
}
```

Runs all test cases

Writing Tests

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

main.cpp

```
#define CATCH_CONFIG_RUNNER
#include "catch.h"

int main(int argc, char **argv) {
  Catch::Session().run(1, argv);
  return 0;
}
```

Use `argc` to let Catch2 have arguments

Writing Tests

sum.cpp

```
#include "catch.h"

int sum(int a[], int n) {
  int i, s = 0;
  for (i = 0; i < n; i++)
     s += a[i];
  return s;
}

TEST_CASE( "sum" ) {
  int a[] = { 1, 2, 3 };
  CHECK( sum(a, 0) == 0 );
  CHECK( sum(a, 3) == 6 );
  CHECK( sum(a, 2) == 3 );
}
```

main.cpp

```
#define CATCH_CONFIG_RUNNER
#include "catch.h"

int main(int argc, char **argv) {
  Catch::Session().run(1, argv);
  return 0;
}
```

Even if you don't use `Catch::Session().run`, exactly one `.cpp` must have `#define CATCH_CONFIG_RUNNER`

Part 2: Classes in C++

Simple Class

```
class Posn {
public:
  int x;
  int y;

  Posn(int x, int y) {
    this->x = x;
    this->y = y;
  }

  virtual bool close_to_origin(int d) {
    return (abs(x) < d) && (abs(y) < d);
  }
};
```

Simple Class

```
class Posn {
public:
  int x;
  int y;

  Posn(int x, int y) {
    this->x = x;
    this->y = y;
  }

  virtual bool close_to_origin(int d) {
    return (abs(x) < d) && (abs(y) < d);
  }
};
```

-> instead of .

Simple Class

```
class Posn {
public:
  int x;
  int y;

  Posn(int x, int y) {
    this->x = x;
    this->y = y;
  }

  virtual bool close_to_origin(int d) {
    return (abs(x) < d) && (abs(y) < d);
  }
};
```

needed to allow override

Separating Declaration and Implementation

```
class Posn {
public:
  int x;
  int y;

  Posn(int x, int y);
  virtual bool close_to_origin(int d);
};


Posn::Posn(int x, int y) {
  this->x = x;
  this->y = y;
}

bool Posn::close_to_origin(int d) {
  return (abs(x) < d) && (abs(y) < d);
}
```

Separating Declaration and Implementation

```
class Posn {
public:
  int x;
  int y;

  Posn(int x, int y);
  virtual bool close_to_origin(int d);
};

Posn::Posn(int x, int y) {
  this->x = x;
  this->y = y;
}

bool Posn::close_to_origin(int d) {
  return (abs(x) < d) && (abs(y) < d);
}
```

connects to the **Posn** declaration

Separating Declaration and Implementation

posn.h

```
class Posn {
public:
  int x;
  int y;

  Posn(int x, int y);
  virtual bool close_to_origin(int d);
};
```

posn.cpp

```
Posn::Posn(int x, int y) {
  this->x = x;
  this->y = y;
}

bool Posn::close_to_origin(int d) {
  return (abs(x) < d) && (abs(y) < d);
}
```

Referring to Classes

```
class Circle {
public:
  Posn* center;
  int radius;

  Circle(Posn* center, int radius) {
    this->center = center;
    this->radius = radius;
  }

  int area() {
    return radius * radius * 3.14;
  }
};
```

Referring to Classes

Use **Posn*** to refer to a **Posn** object

```
class Circle {
public:
  Posn* center;
  int radius;

  Circle(Posn* center, int radius) {
    this->center = center;
    this->radius = radius;
  }

  int area() {
    return radius * radius * 3.14;
  }
};
```

Referring to Classes

```
class Circle {
public:
  Posn* center;
  int r
  Circle(Posn* center, int radius) {
    this->center = center;
    this->radius = radius;
  }

  int area() {
    return radius * radius * 3.14;
  }
};
```

```
new Circle(new Posn(1, -3), 10)
```

Referring to Classes

```
class Circle {
public:
  Posn* center;
  int radius;

  Circle(Posn* center, int radius) {
    this->center = center;
    this->radius = radius;
  }

  int area() {
    return radius * radius * 3.14;
  }
};
```

So far, could declare with

```
class Posn;
```

before `class Circle`...

Referring to Classes

```
class Circle {
public:
  Posn* center;
  int radius;

  Circle(Posn* center, int radius) {
    this->center = center;
    this->radius = radius;
  }

  int area() {
    return radius * radius * 3.14;
  }

  bool covers_origin() {
    return center->close_to_origin(radius);
  }
};
```

Referring to Classes

```
class Circle {
public:
  Posn* center;
  int radius;

  Circle(Posn* center, int radius) {
    this->center = center;
    this->radius = radius;
  }

  int area() {
    return radius * radius * 3.14;
  }

  bool covers_origin() {
    return center->close_to_origin(radius);
  }
};
```

Needs `#include "posn.h"`

Separating Declaration and Implementation

circle.h

```
class Posn;

class Circle {
  public:
    Posn* center;
    int radius;

    Circle(Posn* center, int radius);
    int area();
    bool covers_origin();
  };
```

circle.cpp

```
#include "posn.h"
#include "circle.h"

Circle::Circle(Posn* center, int radius) {
  this->center = center;
  this->radius = radius;
}

int Circle::area() {
  return radius * radius * 3.14;
}

bool Circle::covers_origin() {
  return center->close_to_origin(radius);
}
```

Separating Declaration and Implementation

circle.h

```
class Posn;

class Circle {
  public:
    Posn* center;
    int radius;

    Circle(Posn* center, int radius);
    int area();
    bool covers_origin();
  };
```

Avoid `#include "....h"` in `.h` files

circle.cpp

```
#include "posn.h"
#include "circle.h"

Circle::Circle(Posn* center, int radius) {
  this->center = center;
  this->radius = radius;
}

int Circle::area() {
  return radius * radius * 3.14;
}

bool Circle::covers_origin() {
  return center->close_to_origin(radius);
}
```

Subclasses

posn3d.h

```
#include "posn.h"

class Posn3D : public Posn {
  int z;

  Posn3D(int x, int y, int z);

  virtual bool close_to_origin(int d);
};
```

posn3d.cpp

```
#include "posn3d.h"

Posn3D::Posn3D(int x, int y, int z)
: Posn(x, y)
{
   this->z = z;
}

bool Posn3D::close_to_origin(int d) {
  return (Posn::close_to_origin(d)
          && (abs(z) < d));
}
```

Subclasses

Means **extends**

posn3d.h

```
#include "posn.h"

class Posn3D : public Posn {
  int z;

  Posn3D(int x, int y, int z);

  virtual bool close_to_origin(int d);
};
```

posn3d.cpp

```
#include "posn3d.h"

Posn3D::Posn3D(int x, int y, int z)
: Posn(x, y)
{
   this->z = z;
}

bool Posn3D::close_to_origin(int d) {
  return (Posn::close_to_origin(d)
          && (abs(z) < d));
}
```

Subclasses

Needed for subclassing

posn3d.h

```
#include "posn.h"

class Posn3D : public Posn {
  int z;

  Posn3D(int x, int y, int z);

  virtual bool close_to_origin(int d);
};
```

posn3d.cpp

```
#include "posn3d.h"

Posn3D::Posn3D(int x, int y, int z)
: Posn(x, y)
{
   this->z = z;
}

bool Posn3D::close_to_origin(int d) {
  return (Posn::close_to_origin(d)
          && (abs(z) < d));
}
```

Subclasses

posn3d.h

```
#include "posn.h"

class Posn3D : public Posn {
  int z;

  Posn3D(int x, int y, int z);

  virtual bool close_to_origin(int d);
};
```

posn3d.cpp

```
#include "posn3d.h"

Posn3D::Posn3D(int x, int y, int z)
: Posn(x, y)
{
   this->z = z;
}

bool Posn3D::close_to_origin(int d) {
  return (Posn::close_to_origin(d)
          && (abs(z) < d));
}
```

Superclass constructor

Subclasses

posn3d.h

```
#include "posn.h"

class Posn3D : public Posn {
  int z;

  Posn3D(int x, int y, int z);

  virtual bool close_to_origin(int d);
};
```

posn3d.cpp

```
#include "posn3d.h"

Posn3D::Posn3D(int x, int y, int z)
: Posn(x, y)
{
    this->z = z;
}

bool Posn3D::close_to_origin(int d) {
  return (Posn::close_to_origin(d)
          && (abs(z) < d));
}
```

Super method call

Subclasses

posn3d.h

```
#include "posn.h"

class Posn3D : public Posn {
  int z;

  Posn3D(int x, int y, int z);

  virtual bool close_to_origin(int d);
};
```

Can't avoid `#include "posn.h"` in `posn3d.h`

posn3d.cpp

```
#include "posn3d.h"

Posn3D::Posn3D(int x, int y, int z)
: Posn(x, y)
{
   this->z = z;
}

bool Posn3D::close_to_origin(int d) {
  return (Posn::close_to_origin(d)
          && (abs(z) < d));
}
```

Constructor Convention

When a slide has a class declaration without a constructor

```
class Circle {
 public:
   Posn* center;
   int radius;
};
```

assume a default constructor that takes each field:

```
Circle(Posn* center, int radius) {
  this->center = center;
  this->radius = radius;
}
```

The compiler doesn't know this convention, so you have to type it out!

Equals Convention

When a slide has a class declaration without an `equals` method

```
class Circle {
 public:
   Posn* center;
   int radius;
};
```

assume a implementation that compares each field:

```
virtual bool equals(Circle *other) {
  return (this->center->equals(other->center)
          && this->radius == other->radius);
}
```

The compiler doesn't know this convention, so you have to type it out!

Part 3: Interface-like Classes in C++

Simple Arithmetic Expressions

```
1

2 + 3

4 * 5

6*(7+8)

(1+2) * (3+4*6)
```

Each of those is an *expression*

An *expression* can be nested in another *expression*

To calculate the result, we can ignore details like whitespace and parentheses

Simple Arithmetic Expressions

```
1

2 + 3

4 * 5

6*(7+8)

(1+2) * (3+4*6)
```

Three kinds of *expression*s:

- number
- addition of two expressions
- multiplication of two expressions

Simple Arithmetic Expressions

```
1

2 + 3

4 * 5

6*(7+8)

(1+2) * (3+4*6)
```

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ **+** ⟨expr⟩
| ⟨expr⟩ ***** ⟨expr⟩

Simple Arithmetic Expressions

```
1

2 + 3

4 * 5

6*(7+8)

(1+2) * (3+4*6)
```

Abstract syntax *grammar*

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ **+** ⟨expr⟩
| ⟨expr⟩ ***** ⟨expr⟩

Simple Arithmetic Expressions

```
1

2 + 3

4 * 5

6*(7+8)

(1+2) * (3+4*6)
```

Abstract syntax *grammar*

⟨expr⟩ = ⟨number⟩

"or" | ⟨expr⟩ **+** ⟨expr⟩

| ⟨expr⟩ ***** ⟨expr⟩

Simple Arithmetic Expressions

```
1
2 + 3
4 * 5
6*(7+8)
(1+2) * (3+4*6)
```

Abstract syntax *grammar*

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ **+** ⟨expr⟩
| ⟨expr⟩ ***** ⟨expr⟩

Gray box corresponds to literal text

Simple Arithmetic Expressions

```
1

2 + 3

4 * 5

6*(7+8)

(1+2) * (3+4*6)
```

Abstract syntax *grammar*

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ **+** ⟨expr⟩
| ⟨expr⟩ ***** ⟨expr⟩

⟨expr⟩ corresponds to an interface, and
three classes should implement it

Arithmetic Representation Classes

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ **+** ⟨expr⟩
| ⟨expr⟩ ***** ⟨expr⟩

```
class Expr { };
```

```
class Num : public Expr {
};
```

```
class Add : public Expr {
};
```

```
class Mult : public Expr {
};
```

Arithmetic Representation Classes

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩
| ⟨expr⟩ * ⟨expr⟩

```
class Expr { };
```

```
class Num : public Expr {
public:
  int val;
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;
};
```

Arithmetic Representation Classes

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩
| ⟨expr⟩ * ⟨expr⟩

```
class Expr { };
```

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Arithmetic Representation Classes

```
42

⇒

new Num(42)
```

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ **+** ⟨expr⟩
| ⟨expr⟩ ***** ⟨expr⟩

```
class Expr { };
```

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Arithmetic Representation Classes

```
42+2

⇒

new Add(new Num(42),
        new Num(2));
```

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩
| ⟨expr⟩ * ⟨expr⟩

```
class Expr { };
```

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Arithmetic Representation Classes

```
((42)+2)
```

⇒

```
new Add(new Num(42),
        new Num(2));
```

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩
| ⟨expr⟩ * ⟨expr⟩

```
class Expr { };
```

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Arithmetic Representation Classes

```
((42)+2)
```

⇒

```
new Add(new Num(42),
        new Num(2));
```

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩

For tests, we'll need a way to compare expressions

```
class Expr { };
```

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Arithmetic Representation Classes

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩
| ⟨expr⟩ * ⟨expr⟩

```
class Expr {
public:
  virtual bool equals(Expr *e) = 0;
};
```

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Arithmetic Representation Classes

⟨expr⟩ = ⟨number⟩
| ⟨expr⟩ + ⟨expr⟩
| ⟨expr⟩ * ⟨expr⟩

```
class Expr {
public:
  virtual bool equals(Expr *e) = 0;
};
```

= 0 means each subclass *must* override

```
class Num : public Expr {
public:
  int val;

  Num(int val) {
    this->val = val;
  }
};
```

```
class Add : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Add(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

```
class Mult : public Expr {
public:
  Expr *lhs;
  Expr *rhs;

  Mult(Expr *lhs, Expr *rhs) {
    this->lhs = lhs;
    this->rhs = rhs;
  }
};
```

Implementing Equality

```
bool Class::equals(Interface *o) {
    Class *c = dynamic_cast<Class*>(o);
    if (c == NULL)
      return false;
    else
      return ... check this and c fields equal ...;
}
```

Bolierplate, but easy to get wrong, so make sure you have tests for `equals`

Implementing Equality

```
class Animal {
  virtual bool equals(Animal *o) = 0;
};
```

```
class Tiger : public Animal {
  std::string color;
  int stripe_count;

  bool equals(Animal *a) {
    return ...;
  }
};
```

```
class Snake : public Animal {
  std::string color;
  int weight;
  std::string food;

  bool equals(Animal *a) {
    return ...;
  }
};
```

Implementing Equality

```
class Animal {
  virtual bool equals(Animal *o) = 0;
};
```

```
class Tiger : public Animal {
  std::string color;
  int stripe_count;

  bool equals(Animal *a) {
    Tiger *t = dynamic_cast<Tiger*>(a);
    if (t == NULL)
      return false;
    else
      return (this->color == t->color
              && this->stripe_count == t->stripe_count);
  }
};
```

```
class Snake : public Animal {
  std::string color;
  int weight;
  std::string food;

  bool equals(Animal *a) {
    return ...;
  }
};
```

Implementing Equality

```
class Animal {
  virtual bool equals(Animal *o) = 0;
};
```

```
class Tiger : public Animal {
  std::string color;
  int stripe_count;

  bool equals(Animal *a) {
    return ...;
  }
};
```

```
class Snake : public Animal {
  std::string color;
  int weight;
  std::string food;

  bool equals(Animal *a) {
    Snake *s = dynamic_cast<Snake*>(a);
    if (s == NULL)
      return false;
    else
      return (this->color == s->color
              && this->weight == s->weight
              && this->food == s->food);
  }
};
```