# Big C++: Polymorphism

## Poly what now?

- Objects that adhere to a standard set of properties and behaviors can be used interchangeably.
- Implemented by Overloading and Overriding

## Why bother?

- Avoid code duplication/reuse where not necessary
- Simplifies and structures code
- Common interface
- Consistency of design should be more understandable
- Debugging

SciNet compute • calcul CANADA

# Operator Overloading

- Use expected syntax for non-built in Types
- A = B + C, regardless of what A, B, or C is.

## Syntax

### Declaration

```
classname {
      ...
   public:
      classname& operator=(classname & anobject);
      ...
};
```

### Definition

```
classname& classname::operator=(classname & anobject) {
   statements
   return *this;
}
```

# Operator Overloading

## Example (Matrix Class)

```
class matrix {
   private:
      int rows, cols;
      double *elements;
   public:
      matrix(int r, int c);
      ~matrix();
      matrix& operator= (matrix &m);
      int get_rows();
      int get_cols();
      void fill(double value);
      matrix operator+ (const matrix &C);
};
```

# Operator Overloading

## Example (Add two martices)

```
matrix A(5,5), B(5,5), C(5,5);
A.fill(1.0); B.fill(1.0); C.fill(1.0);
for (int i=0, i<row; i++)
   for (int j=0, j<cols; j++)
      A[i][j] = B[i][j] + C[i][j];
```

## Example (Add two martices using "+" operator)

```
matrix A(5,5), B(5,5), C(5,5);
A.fill(1.0); B.fill(1.0); C.fill(1.0);
A = B + C;
```

# Operator Overloading

## "+" Operator

```
matrix matrix::operator+ (const matrix &C) {
   matrix Temp(*this);
   for (int i=0, i<rows*cols; i++)
      Temp.elements[i] += C.elements[i];
   return Temp;
};
```

## "+=" Operator

```
matrix& matrix::operator+= (const matrix &C) {
   for (int i=0, i<rows*cols; i++)
      elements[i] += C.elements[i];
};
```

compute • calcul CANADA

# Operator Overloading

## const

- set a constant variable at compile time
- keyword to protect your variables
- const references

## "+=" Operator with bounds checking

```
matrix& matrix::operator+= (const matrix &C) {
   if ( rows == C.rows && cols == C.cols) {
      for (int i=0, i<rows*cols; i++)
         elements[i] += C.elements[i];
   } else {
      cerr<<"Matrix Indicies don't match, can't add";
      exit(1);
   }
};
```

# Operator Overloading

## "( )" Operator

```
double & matrix::operator() (int &i, int &j) {
      return elements[i*cols + j];
};
```

```
A(1,4) = 6;
double y = A(1,4);
```

## "[ ]" Index Operator

```
double matrix::operator[] (int &i) {
      return elements[i];
};
```

# Operator Overloading

## "<<" ">>" Stream Operators

```
std::ostream& operator << (std::ostream& o, matrix& m) {
   for (int i=0; i<m.get_rows() ; i++) {
      for (int j=0; j<m.get_cols(); j++) {
         o << m(i,j) << " ";
      }
      o << std::endl;
   }
};
```

```
   std::cout<<"Matrix A = "<< A << std::endl;
```

# Operator Overloading

## Friends

- friend keyword allows non-member functions access to private data.

## "<<" ">>" Stream Operator using friend

```
class matrix {
 ...
 friend std::ostream& operator<<(std::ostream& o, matrix& m);
};
```

```
std::ostream& operator<<(std::ostream& o, matrix& m) {
   for (int i=0; i<rows ; i++) {
      for (int j=0; j<cols; j++) {
         o << elements[i*cols + j] << " ";
      }
      o << std::endl;
   }
};
```

# Operator Overloading

## C++ operators available to overload

| | | | | | | |
|---|---|---|---|---|---|---|
| `+` | `-` | `*` | `/` | `%` | `^` | `&` |
| `\|` | `~` | `!` | `=` | `<` | `>` | `+=` |
| `-=` | `*=` | `/=` | `%=` | `^=` | `&=` | `\|` |
| `<<` | `>>` | `>> =` | `<< =` | `==` | `!=` | `<=` |
| `>=` | `&&` | `\|\|` | `++` | `--` | `->*` | `,` |
| `->` | `[ ]` | `( )` | `new` | `new[ ]` | `delete` | `delete[ ]` |

## HANDS-ON:

Overload **+=**, **()**, and stream operators for matrix c++ class, and rewrite main to use it.