Practice Sessions
Astrophysical Simulations

Part 5: C++ classes and objects

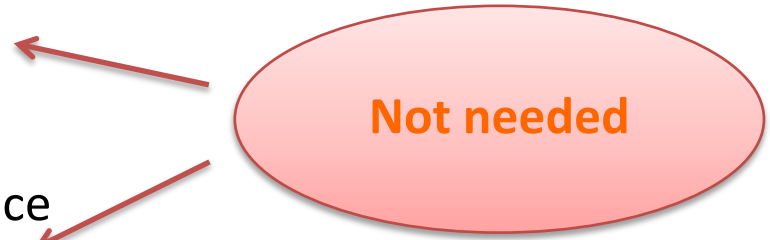
GHENT
UNIVERSITY

Master of Science in Physics and Astronomy
2018-2019
Peter Camps
peter.camps@ugent.be
S9, 1st floor, office 110.014

C++ classes

Object-oriented programming

- Abstraction – define interfaces for specific domain concepts
  - » Standard operators can be overloaded for user-defined types
- Encapsulation – ensure data integrity by controlling access
  - » Allocate and release memory in a safe way (construct/destruct)
- Inheritance – acquire properties of other data types, enabling reuse and modular design of complex systems
- Polymorphism – provide common interface for multiple implementations, allowing components to act differently



Although these are key features for larger software designs

Data members, constructors and getters

```
// represents a 2-D vector
class Vec {

    double _x;
    double _y;

public:
    Vec() { _x=0; _y=0; }
    Vec(double x, double y)
          { _x = x; _y = y; }

    double x() const { return _x; }
    double y() const { return _y; }

    . . .
};
```

Declare a user-defined data type (or class); by convention the name starts with a capital

Initial items are private, i.e. can be accessed only from within the class

Declare data members; each object (instance) of this class will have a copy; by convention the name starts with an _

Subsequent items can be accessed from outside the class

Constructor has class name and can take arguments to initialize data members

Getter provides read-only access to the data member with the corresponding name

Semicolon is required here

Indicates to the compiler that the function does not change any data members

Using instances of a class

```
int main()
{
    Vec a(3,4);
    double norm = hypot(a.x(), a.y());
    cout << norm << endl;

    Vec b;
    cout << b.x() << ' ' << b.y() << endl;

    b = a;
    cout << b.x() << ' ' << b.y() << endl;

    b = Vec(-1.5, 6.7);
    cout << b.x() << ' ' << b.y() << endl;
}
```

Declare and construct Vec object with given x, y components

Use getters in expression

Default constructor initializes to null vector

Assignment copies all data members of object

Construct temporary Vec object and assign it

```
5
0 0
3 4
-1.5 6.7
```

Output of the program

Other member functions

```
class Vec {
    public:

    . . .

    double norm() const
        { return sqrt(_x*_x + _y*_y); }

    double norm2() const
        { return _x*_x + _y*_y; }

    double norm3() const
        { double r = sqrt(_x*_x + _y*_y); return r*r*r; }

    . . .
};
```

Operations that involve the data members of the class (or rather, the object) can be provided as member functions of the class (or rather, the object)

Indicates to the compiler that the function does not change any data members

One can also define operations that update the data members

Operator overloading – member functions

```
class Vec {
    public:

    . . .

        Vec& operator+=(Vec v) {
            _x += v._x;
            _y += v._y;
            return *this;
        }

        Vec& operator*=(double s) {
            _x *= s;
            _y *= s;
            return *this;
        }

    . . .
};
```

C++ allows redefining (overloading) most built-in operators (including +,-,*,/) for user-defined data types

Overload the += assignment operator to add the specified vector to the vector for which the function is being invoked; use similar code for overloading the -= operator

"reference to Vec"

"this object"

Overload the *= assignment operator to multiply the vector for which the function is being invoked by a scalar; use similar code for overloading the /= operator

Operator functions must return a reference to the object under consideration, which is accomplished through the mechanism shown in the example

Operator overloading – out-of-class

```
class Vec {
    • • •
};
```

The regular binary operators don't need access to the private data members because they can be defined in function of the assignment operators, so they can be provided outside of the class definition

```
// operations between two vectors
Vec operator+(Vec a, Vec b) { return a += b; }
Vec operator-(Vec a, Vec b) { return a -= b; }
```

Use one of the arguments to store the result of the operation (using an operator defined in-class) and then return the result (the in class operators return a reference to the object itself)

```
// operations between a vector and a scalar
Vec operator*(Vec a, double s) { return a *= s; }
Vec operator*(double s, Vec b) { return b *= s; }
Vec operator/(Vec a, double s) { return a /= s; }
```

The binary operation "scalar * vector" has to be provided out-of-class because in-class operators always have the receiving object as the left-hand-side operand

Using overloaded operators

```
void print(Vec a)
    { cout << a.x() << ' ' << a.y() << endl; }

int main()
{
    double s = 3;
    Vec a(2,3);
    Vec b(4,5);

    print(a+b);
    print(s*a);
    print(s*a - s*s*b/2);
    a += s*b;
    print(a);
    b /= b.norm();
    print(b);
}
```

```
6 8
6 9
-12 -13.5
14 18
0.624695 0.780869
```

Output of the program

Questions?