

# Chapter 6. Struct and Classes

Programming Concepts in Scientific  
Programming

EPFL, Master class

October 27, 2019

# Types

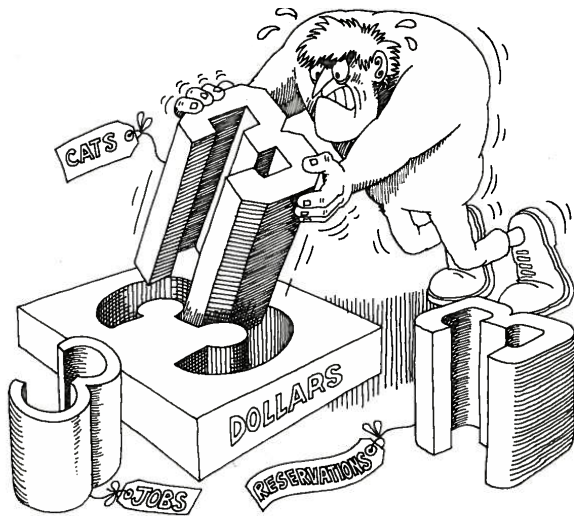
## Known types

- ▶ void
- ▶ char
- ▶ short
- ▶ int
- ▶ long int
- ▶ float
- ▶ double
- ▶ long double
- ▶ pointers

## Specifiers

- ▶ unsigned
- ▶ const

# Types



How to define new types ?

# New types

How ?

```
struct NameType {  
    double a;  
    int b;  
};
```

# New types

How ?

```
struct NameType {  
    double a;  
    int b;  
};
```

# New types

How ?

```
struct NameType {  
    double a;  
    int b;  
};
```

# New types

How ?

```
struct NameType {  
    double a;  
    int b;  
};
```



# Structures

## Example

### Definition

```
struct Planet {  
    double coords[3];  
    std::string name;  
};
```

# Structures

## Example

### Definition

```
struct Planet {  
    double coords[3];  
    std::string name;  
};
```

### Creating a variable

```
Planet p;
```

# Structures

## Example

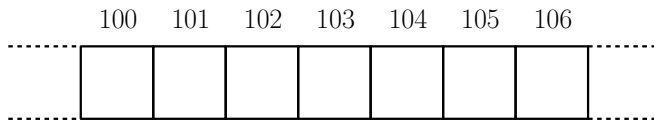
### Definition

```
struct Planet {  
    double coords[3];  
    std::string name;  
};
```

### Creating a variable

```
Planet p;
```

### How is the memory representation ?



# Structures

## Example

How to access the variables of a structure

```
Planet p;  
p.coords[0] = 10;
```

# Structures

## Example

How to access the variables of a structure

```
Planet p;  
p.coords[0] = 10;
```

How to get the size of a structure (in bytes)

```
std::cout << sizeof(Planet) << std::endl;
```

What is a class ?

What is a class ?

A type associating  
**Data** and **Functions**

A blue print to fabricate **Objects**

What is an object ?



What is an object ?

An **instanciation**(variable)  
of a **class/struct** type

# Classes

## Gathering data and functions

```
struct Planet {  
    void move(double delta[3]);  
    double coords[3];  
};
```

# Classes

## Gathering data and functions

```
struct Planet {  
    void move(double delta[3]);  
    double coords[3];  
};
```

### Vocabulary

- ▶ variables (state): members
- ▶ functions: methods

# Classes

## Gathering data and functions

```
struct Planet {  
    void move(double delta[3]);  
    double coords[3];  
};
```

### Vocabulary

- ▶ variables (state): members
- ▶ functions: methods

### Usage:

```
p.move(delta);
```

Can be done in C with  
multiple files (modules) ?

```
Planet p;  
p.move(delta);
```

# Classes

## Encapsulation

```
class Planet {  
  
public:  
    Planet();                // constructor  
    ~Planet();              // destructor  
    void move(double delta[3]); // a method  
  
private:  
    double coords[3]; // a member  
};
```

# Classes

## Encapsulation

```
class Planet {  
  
public:  
    Planet();           // constructor  
    ~Planet();         // destructor  
    void move(double delta[3]); // a method  
  
private:  
    double coords[3]; // a member  
};
```

# Classes

## Encapsulation

```
class Planet {  
  
    public:  
        Planet();                // constructor  
        ~Planet();              // destructor  
        void move(double delta[3]); // a method  
  
    private:  
        double coords[3]; // a member  
};
```



# Classes

## Encapsulation

```
class Planet {  
  
public:  
    Planet();           // constructor  
    ~Planet();         // destructor  
    void move(double delta[3]); // a method  
  
private:  
    double coords[3]; // a member  
};
```

- ▶ It is an interface (declaration in a .hh/.hpp file)

# Classes

## Encapsulation

```
class Planet {  
  
public:  
    Planet();                // constructor  
    ~Planet();              // destructor  
    void move(double delta[3]); // a method  
  
private:  
    double coords[3]; // a member  
};
```

- ▶ It is an interface (declaration in a .hh/.hpp file)
- ▶ Methods and members are accessible/inaccessible

# Classes

Methods definitions (.cpp/.cpp)

```
#include "planet.hh"
```

```
void Planet::move(double delta[3]) {  
    // DO SOME CODE  
}
```

# Classes

Methods definitions (.cpp/.cpp)

```
#include "planet.hh"

void Planet::move(double delta[3]) {
    // DO SOME CODE
}
```

# Classes

## Construction/Destruction

Constructor: set the initial state

```
Planet::Planet() {  
    coords[0] = 0.;  
    coords[1] = 1.;  
    coords[2] = 2.;  
}
```

# Classes

## Construction/Destruction

Constructor: set the initial state

```
Planet::Planet() {  
    coords[0] = 0.;  
    coords[1] = 1.;  
    coords[2] = 2.;  
}
```

Destructor

```
Planet::~~Planet() {}
```

# Classes

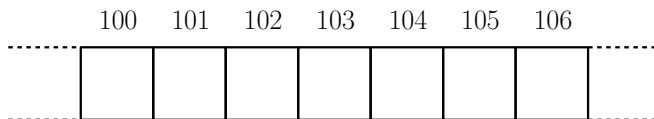
## Construction/Destruction

Constructor: set the initial state

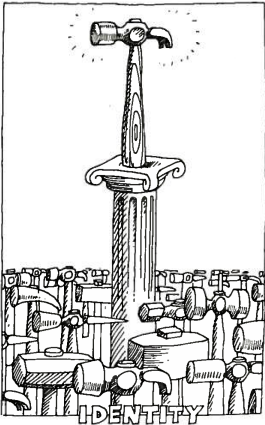
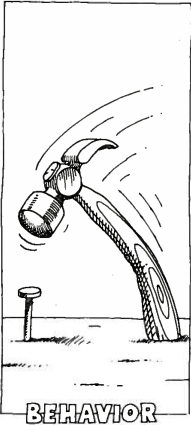
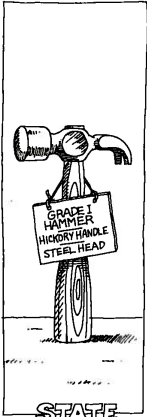
```
Planet::Planet() {  
    coords[0] = 0.;  
    coords[1] = 1.;  
    coords[2] = 2.;  
}
```

Destructor

```
Planet::~~Planet() {}
```



# Classes & Objects





# Classes

## Constructor/Destructor

- ▶ Constructor sets the initial state
- ▶ Destructors release memory allocations

```
class Planet {  
  
public:  
    Planet();                // constructor  
    ~Planet();              // destructor  
    void move(double delta[3]); // a method  
  
private:  
    // a pointer member  
    double *coords;  
};
```

# Classes

## Constructor/Destructor

```
Planet::Planet() {  
    coords = new double[3];  
    coords[0] = 0.;  
    coords[1] = 1.;  
    coords[2] = 2.;  
}
```

```
Planet::~~Planet() {  
    // delete memory  
    delete[] coords;  
}
```

# Classes

## Constructor with parameter(s)

```
class Planet {  
  
public:  
    Planet(); // constructor  
    Planet(double param1, int param2);  
    Planet(const Planet &); // copy constructor  
    ~Planet(); // destructor  
    void move(double delta[3]); // a method  
  
private:  
    // a pointer member  
    double *coords;  
};
```

# Classes

## Constructor with parameter(s)

```
class Planet {  
  
public:  
    Planet(); // constructor  
    Planet(double param1, int param2); // second constructor  
    Planet(const Planet &); // copy constructor  
    ~Planet(); // destructor  
    void move(double delta[3]); // a method  
  
private:  
    // a pointer member  
    double *coords;  
};
```

# Classes

## Copy constructor

```
Planet::Planet(const Planet &p) {
```

```
    // copy pointer ?
```

```
    coords = p.coords;
```

# Classes

## Copy constructor

```
Planet::Planet(const Planet &p) {
```

```
    // copy pointer ?
```

```
    coords = p.coords;
```

```
    // or copy the content ?
```

```
    coords = new double[3];
```

```
    for (int i = 0; i < 3; ++i) {
```

```
        coords[i] = p.coords[i];
```

```
    }
```

```
}
```

# Objects

Pointer/reference to object

```
Planet p;
```

```
Planet *ptr = &p;
```

```
Planet &ref = p;
```

## Objects

Pointer/reference to object

```
Planet p;
```

```
Planet *ptr = &p;
```

```
Planet &ref = p;
```

Dynamically allocate an object

```
Planet *p1 = new Planet; // no parentheses!
```

```
Planet *p2 = new Planet(param1, param2);
```



## Objects

Pointer/reference to object

```
Planet p;  
Planet *ptr = &p;  
Planet &ref = p;
```

Dynamically allocate an object

```
Planet *p1 = new Planet; // no parentheses!  
Planet *p2 = new Planet(param1, param2);  
  
p1->move(coords);
```

# Objects

this pointer

What is the mysterious **this** ?

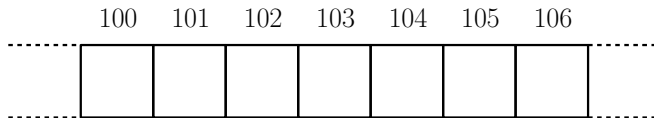
```
struct Planet {  
  
    void test() { std::cout << this << std::endl; }  
};  
  
int main() {  
    Planet p;  
    std::cout << &p << std::endl;  
    p.test();  
}
```

# Objects

this pointer

What is the mysterious **this** ?

```
struct Planet {  
  
    void test() { std::cout << this << std::endl; }  
};  
  
int main() {  
    Planet p;  
    std::cout << &p << std::endl;  
    p.test();  
}
```



## Classes friends

What happens if we do this ? (try it)

```
class A {
```

```
private:
```

```
    int secret;  
};
```

```
class B {
```

```
    int getSecret(A &a) { return a.secret; }  
};
```

## Classes friends

```
class A {  
  
private:  
    friend B;  
    int secret;  
};  
  
class B {  
    int getSecret(A &a) { return a.secret; }  
};
```

## Classes friends

```
class A {  
  
private:  
    friend int toto(A &a);  
    int secret;  
};  
  
int toto(A &a) { return a.secret; }
```

## Class operators

```
class A {  
public:  
    int operator[](int i) {  
        // modifies the behavior  
        return values[i] * 2;  
    }  
};
```

```
private:  
    int values[100];  
};
```

## Class operators

```
class A {  
public:  
    int operator[] (int i) {  
        // modifies the behavior  
        return values[i] * 2;  
    }  
  
private:  
    int values[100];  
};
```



## Class operators

```
class A {  
public:  
    int operator[] (int i) {  
        // modifies the behavior  
        return values[i] * 2;  
    }  
  
private:  
    int values[100];  
};
```

You defined the operator [.]

```
std::cout << a [2] << std::endl;
```

## Const in methods

```
class A {  
public:  
    int doItConst() const {  
  
        values[0] = 2; // error: cannot touch the members  
  
        return values[0];  
    }  
  
private:  
    int values[100];  
};
```

# Class

## Take away message

- ▶ **Class:** A type associating **Data** and **Functions**
- ▶ **Object/Instance:** A variable of a **class/struct** type
- ▶ **Methods:** Functions in a class
- ▶ **Members:** Variables in a class
- ▶ **Encapsulation:** mechanism allowing **public** and **private** sections
- ▶ **Operators:** special functions to define operators `()[]*-/+` etc.
- ▶ **this:** pointer to current object