2 Destructors

A destructor is a special member function that is called when the lifetime of an object ends. The purpose of the destructor is to free the resources that the object may have acquired during its lifetime.

2.1 Dynamic Array as class member

- For some reason if you want to define array members dynamically,

```cpp
class Data
{
public:
        // constructor
        Data();
    // mutator for time
    void set_time(int n, int ti);
    // accessor for time
    int get_time(int n);
private:
    // size of array (note staic const to define a constant)
    static const int size = 10;
    double *time;
    int distance;
};

Data::Data()
{
        time = new double[size];
}
void Data::set_time(int n, int ti)
{
    if (n >= 0 && n < size){
        time[n] = ti;
    }
}
int Data::get_time(int n)
{
    if (n >= 0 && n < size){
        return time[n];
    }
    return -1;
}
```

  - But how do we free and memory for the array “time”?
    - You know:
      - Dynamic variables do not "go away" unless delete is called
      - Even if a local pointer variable goes away at the end of a function, the dynamic variable it pointed to remains unless delete is called

2.2 Destructor

- A destructor is a member function that is called automatically when an object of the class goes out of scope
  - The destructor contains code to delete all dynamic variables created by the object
  - A class has **only one destructor** with **no arguments**
  - The **name of the destructor** is distinguished from the default constructor by **the tilde symbol ~**

* Example:

```
~Data();
```

2.2.1 ~Data()

- The destructor in the **Data** class must call delete [ ] to return the memory of any dynamic variables to the free-store

```
Data::~Data()
{
        delete [] time;
}
```

2.2.2 When Destructor called?

- When the object is destroyed

```
for (int i = 0 ; i < 10 ; i++)
{
        TimeOfDay time_point;
        :
}// destructor called
```

- If dynamically created,

```
delete [] time_point;  // destructor called
```