

Class

Review: Array

Students in a class:

Mary

John

Eric

Katy

Tim

Review: Parallel Array

Students in a class:

names

0

1

2

3

4

Mary

John

Eric

Katy

Tim

IDs

1111

1121

1234

3214

1232

kpI

9

8

7


6

5

Class


Employee:

e3




Name
ID
KPI

e1



Name
ID
KPI

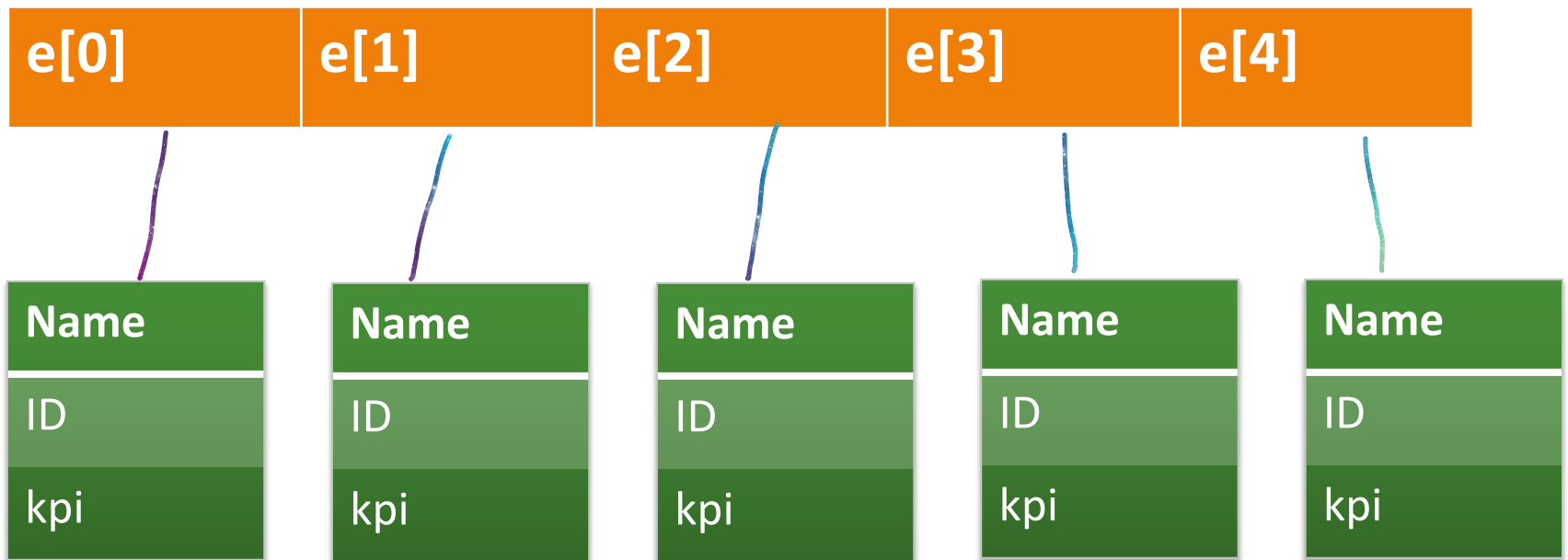
e2



Name
ID
KPI

Class

Employee in a class:



Structured data

- Parallel arrays aren't a natural fit for *heterogeneous* rows of data
 - One set of names, one set of ID, one set of GPAs
- What we have is structured data
 - Name, ID, GPA for each employee
 - One set of employees
- For a single employee we could do:

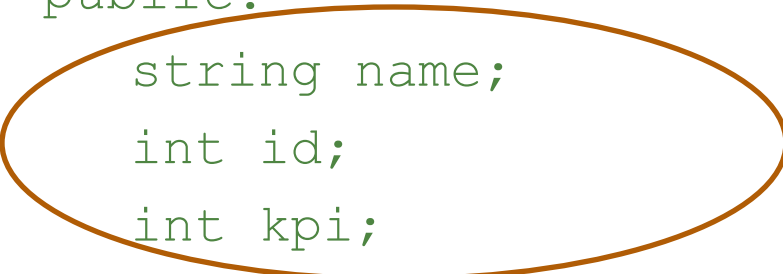
```
string name;  
double id;  
double kpi;
```

- Allocates memory space for 1 strings and 2 double

Using classes

- C++ provides **classes** to group structured data together

```
class Employee
{
    public:
    string name;
    int id;
    int kpi;
};
```



- This is a **class definition**
 - Give the class a name - Employee
 - Tell the compiler what the parts of the class are
 - Each part has a type and a name (looks just like a variable)
 - The parts of a class are called **members**

Using classes

- C++ provides *classes* to group structured data together

```
class Employee
{
    public:
        string name;
        int id;
        int kpi;
};
```


Using classes

- Defining the class creates a blueprint
 - No memory is allocated yet
 - The class is used as a data type in a variable declaration:

- Variable declaration is always:

```
type name;
```

- So in this example:

```
int num;
```

```
num = 10;
```

```
Employee e1;
```

```
e1.name = "peter";
```

- This variable declaration:
 - Allocates memory space for an *instance* of the class
 - 2 strings, 1 int
 - Names that memory space
 - A class instance is also called an [object](#)

Using class objects

- With arrays, you always have to indicate which element in the array you want to use
 - Using the array subscript operator []
 - E.g. `this_array[15]`
- With class objects, you have to indicate which part of the class you want to use
 - The *member access operator* (.) indicates part of an object
 - The parts are used like any other variable:

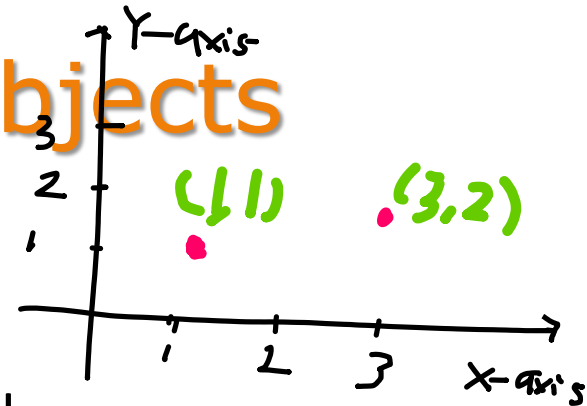
```
emp.name = "peter";  
cin >> emp.id;  
emp.kpi= emp.kpi + 1;
```

Arrays of objects

- Now that we've defined a class for employee
 - We can have a set of employees using an array

```
employee e[10];
```
 - Allocates space for 10 employee objects
 - Each one has 2 strings and 1 int
- Combine array and class access operators
 - The 6th employee's name:
 - `e[5].name`
 - the first employee's review score:
 - `e[0].kpi = 10;`
 - the Id of the 5th employee
 - `cout << e[4].id << endl;`

Exercise: arrays of objects



- Define a class to hold a point (x, y)
 - Like you would use to specify points on the screen
- Write a statement to declare an array of 100 points
- Write statements to set the first point to (1, 4)
 - That is, x is 1, y is 4
- Write statements to set the second point to (5, 3)
- Assuming there is an integer n, and there are n valid points in your array:
 - Write statements to print the values of all 100 points to the screen

Example: lookup a record

- Given the arrays of employee objects and the following code:

```
string lookup_name;  
cout << "Enter a name to look up: ";  
cin >> lookup_name;
```

- **Write a function** to return the requested employee
- Write a function to print an employee object
 - E.g. “samir (developer) received a review of 75”

Exercise: lookup the highest score

- Given the array of employee objects
- Write a function to return the employee object with the highest review score