Destructor

- C++ allows you to execute a piece of code when an object is destroyed.
- This is very useful to release some resources (e.g. Heap Memory) that have been allocated in your constructor.
    - Like constructor, destructor is a public method with no return type.
    - Syntax: `~ClassName()`

Destructor

```
class Matrix{
    int numrows;
    int numcols;
    int *elements;

public:
    Matrix(int nrows, int ncols){
        numrows = nrows;
        numcols = ncols;
        //allocating heap memory for the matrix!
        printf("creating matrix...\n");
        elements = (int*) malloc(nrows*ncols*sizeof(int));
    }
    ~Matrix(){
        //memory will be freed when
        //this matrix object is destroyed.
        printf("freeing matrix...\n");
        free(elements);
    }
};
```

Destructor

```
int main(){
    //create a 2 by 2 matrix
    Matrix m(2,2);
    // do some matrix stuff...
    printf("doing matrix stuff\n");
    return 0;
}
```

The output of the program:

```
creating matrix...
doing matrix stuff
freeing matrix...
```

Although I never explicitly called `~Matrix()`, it has been automatically called before my program exits.

Lifespan of an object

- The lifespan of an object is a complicated topic in C++.
- We only need to remember a few things:
  - When your program finishes, all the objects you have created in the **stack memory** will be automatically destroyed.
  - In the same function, objects will be destroyed in the opposite order they are created.
  - An object created **in the stack memory** of a function will be destroyed when the function exits unless it is the return value.
  - An object created in the **heap memory** will not be destroyed until it is manually freed by the programer.

Creating/Deleting Objects in Heap Memory

- In C++, you can directly create objects in heap memory using the `new` keyword.
- They have to be manually destroyed using the `delete` keyword.

```
//create a matrix object in the heap memory
Matrix *pm = new Matrix(2,2);
//now, pm is a pointer pointing to the matrix
//now do matrix stuff... before you go
delete pm;
//the heap memory can be released by delete
//keyword, this will trigger pm's destructor.
```

Problems of Structure in C

1. Code is poorly reused, which leads to redundancy and confusion.
2. Does not reflect proper hierarchies of data
3. Data and operations on data are detached.
    - Solved! Classes combines variables and functions.

We will see how **the first two problems** are solved using OOP!

Inheritance

- Inheritance is a relationship you can declare between classes and is the way OOP reuses code.
- Inheritance expresses "is-a" relationship between classes.
  - `CSstudent` is a `student`.
  - `sportscar` is a `car`.
  - `diagonalmatrix` is a `matrix`.
- The **Child Class** that inherits from the **Parent Class** will inherit all the code from the parent class.
- In other words, **Child Class** reuses code from **Parent Class**.

Inheritance

Consider the following `student` class:

```
class student{
    int ID;
    char* name;
    int grade;

public:
    void set_grade(int score){
        if(score <= 100 && score > 0){
            grade = score;
        }
    }
    int get_grade(){
        return grade;
    }
};
```

Inheritance

- Now, We want to create a `CSstudent` class.
- We have **two goals**:
  - We want to do so without duplicating the code. i.e., rewriting everything we wrote for `student` class.
  - Our existing code written for `student` should work as is.

(Public) Inheritance Syntax:

```
class child: public parent{

};
```

Child Class

- Create `CSstudent` as a child class of `student`.

```
class CSstudent: public student{

};
```

- Now, the `CSstudent` class has **inherited** all fields and methods of the `student` class. It can do whatever `student` class can do.

```
CSstudent song;
song.set_grade(70);
printf("%d", song.get_grade()); //prints 70.
```

- Inheritance reuses my old code for `student` class, and reduces the redundancy of my code.

Child Class

- You can define fields and methods that are **exclusive** to `CSstudent`.

```
class CSstudent: public student{
    int programming_score;
public:
    int get_programming_score(){
        return programming_score;
    }
    void set_programming_score(int score){
        if(score <= 100 && score > 0){
            programming_score = score;
        }
    }
};
```

Child Class

- Now, **in addition to** all fields and methods that are already in `student`, `CSstudent` has an extra field `programming_score` and two extra methods `get_programming_score` and `set_programming_score`.
- For example:

```
CSstudent song;
song.set_grade(70);
printf("%d\n", song.get_grade());
//prints out 70
song.set_programming_score(80);
//prints out 80.
printf("%d\n", song.get_programming_score());
```

Child Class

- Moreover, all functions that take a `student` object as an input will now take `CSstudent` as input.
    - Since the C++ knows, `CSstudent` is a `student`.
- Suppose we have a function:

```
void print_grade(student s){
    printf("%d\n",s.get_grade());
}
```

- Now we can call `print_grade` using `song` as an input:

```
CSstudent song;
song.set_grade(70);
print_grade(song);
//OK, C++ knows song is a CSstudent,
// thus it deduces that song is a student
//prints 70.
```

Constructors in Inheritance

- Constructors are not inherited!
- You need to rewrite constructors for child classes!
    - Even if the parent classes already have one!

```
class student{
    int ID;
    const char* name;
    int grade;

public:
    student(int newid, const char* newname, int newgrade){
        ID = newid;
        name = newname;
        set_grade(newgrade);
    }

    // rest is omitted
};
```

Constructors in Inheritance

Then you creat a `CSstudent` class inherits from `student`

```
class CSstudent: public student{

};
```

The following does not work!

```
CSstudent jack(1234, "Song", 70); //compilation error!
```

The constructor is not inherited!!
Compiler cannot find a suitable constructor for the given list of input arguments!!

Constructors in Inheritance

You can rewrite a constructor for your new class:

```
class CSstudent: public student{
    int programming_score;

public:
    CSstudent(int newid, const char* newname,
              int newgrade, int p_score)
        :student(newid, newname, newgrade)
        // calling the constructor in the parent class
    {
        programming_score = p_score;
        // validity checking omitted
    }

};
```

Now you can write:

```
CSstudent jack(1234, "Song", 70, 90); // OK!
```

Other Issues in Inheritance

- This lecture only aims to give you a glimpse of OOP in C++. There are many other subjects you need to master before writing a large scale OOP program.
- If you are interested,
- Read here for more information about inheritance.

Conclusion

- Structure in C has some issues:
- PP: You divide your program into sub-procedures.
- OOP: You divide your program into small "objects".
    - Objects contains "fields" and "methods".
- C++
    - It is a superset of C.
    - Class and Objects
    - Fields and Methods
    - Constructor/Destructor
    - Inheritance

Homework 1

1. You will need the matrix class you wrote in the previous lab.
2. You can check your answer with the solution.

Homework 1

1. First, modify your **constructor** and add a **destructor** so that
    - Your `matrix` class manages its own memory.
    - The constructor take only **two input arguments**: `nrow` and `ncol`. It allocates heap memory for your matrix.
    - The destructor releases the heap memory allocated for your matrix.
    - If you have trouble, please read the "Destructor" slide to see how it is done.
2. Write two public methods in the class:
    - `int get_num_rows()` and `int get_num_cols()` returns number of rows and columns of the matrix.

Homework 2 (submit)

1. Images are essentially matrices stored in the memory, so you can say, an image "is a" matrix.
2. Create an `image` class that inherits from the `matrix` class.

Homework 3 (submit)

4. Write an constructor for the `image` class:
    - `image(int height, int weight, int data[])`
    - where `width` and `height` are the width and height of the image, `data` array stores a row-major matrix, representing an image.
    - In the constructor, you need to initialize **all fields** in the image class appropriately.
    - You need to copy the data from `data` to `elements`.
5. Write a method `show()` in the `image` class:
    - It prints the image out to the screen.
    - See the lab in https://github.com/anewgithubname/MATH10017-2022/blob/main/lecs/lec8.pdf

Homework 3 (submit)

1. Write code in `main` and test your `image` class.
2. Does your program have memory leak issue?
    - Hint, print out a message each time you allocate the memory.
    - print out a message each time you free the memory.
    - Do you see the same number of messages for allocating the memory and freeing the memory?

Homework 4 (Challenge)

1. Let us expand our matrix class further, by including a `dot` function tha performs matrix multiplication.
    - Write a public method `matrix dot(matrix B)`.
    - `matrix C = A.dot(B)` will perform the following operation:
    - $C \leftarrow A \cdot B$, here $\cdot$ means matrix multiplication.
2. Test your `dot` function in the main function.
3. Does your code work? What do you see?

Homeworks 4 (Challenge)

4. Your code will trigger a runtime error.
5. `matrix dot(matrix B)` passes B by value, which means, all variables in `B` are **copied**, that includes `B.elements`.
6. Now, there are two `B`s! The copy of `B` in your dot function, and the original `B` outside of the `dot` function.
7. When `dot` function exits, it triggers the copy `B`'s destructor, freeing memory `B.elements` points to.
8. When `main` function exists, it triggers the original `B`'s destruction, and the program tries to free the same memory the second time!
9. Kaboom! Your program crashes.

Homeworks 4 (Challenge)

1. There are several solutions to this problem. The most reliable solution is called **copy constructor**.
2. The copy constructor is a special constructor, triggered automatically when a copy of an object is made (when `B` is passed by value).
3. If no copy constructor is provided (like in our earlier situation), C++ will copy all variables in an object to the new object. This is not good enough as we have seen.
4. We need a "deep copy": Not only we need to copy all variables, but also the memory they point to!!

Homeworks 4 (Challenge)

1. A copy constructor of our matrix takes the following form:

```
matrix(const matrix &M){
    // do deep copy
    // not only copy variables,
    // but also copy the memory they point to.
}
```

2. Don't mind const and & before M, just treat M as if it is a regular variable.
3. Write a copy constructor in the matrix class, does your code now work without any issue?
4. You can read more here and here.