# EXAMPLE 9:

## RULE #2: USE `new/delete` FOR SINGLE VARIABLES OF ALL TYPES.

# RULE #2: USE `new/delete` FOR SINGLE VARIABLES OF ALL TYPES.

ANYTIME YOU NEED TO CREATE A SINGLE VARIABLE OF A POINTER TYPE, JUST `new` TO BOTH ALLOCATE AND CONSTRUCT THE VARIABLE

YOU CAN USE THIS WITH SIMPLE TYPES (INT/FLOAT ETC) AS WELL AS WITH OBJECTS

YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

REFRESHER

# YOU CAN USE THIS WITH SIMPLE TYPES (INT/FLOAT ETC) AS WELL AS WITH OBJECTS

```
float * floatDynamic = new float(23.3);
cout << "Dynamically assigned float has value = " << *floatDynamic << endl;
delete floatDynamic;
```

```
[Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example9.cpp
[Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
 Dynamically assigned float has value = 23.3
```

# YOU CAN USE THIS WITH SIMPLE TYPES (INT/FLOAT ETC) AS WELL AS WITH OBJECTS

```
float * floatDynamic = new float(23.3);
cout << "Dynamically assigned float has value = " << *floatDynamic << endl;
delete floatDynamic;
```

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example9.cpp
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
Dynamically assigned float has value = 23.3
```

# YOU CAN USE THIS WITH SIMPLE TYPES (INT/FLOAT ETC) AS WELL AS WITH OBJECTS

```
float * floatDynamic = new float(23.3);
cout << "Dynamically assigned float has value = " << *floatDynamic << endl;
delete floatDynamic;
```

```
[Vitthals-MacBook-Pro:~ vitthalsrinivasan$ g++ -Wall Example9.cpp
[Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
 Dynamically assigned float has value = 23.3
```

# RULE #2: USE `new/delete` FOR SINGLE VARIABLES OF ALL TYPES.

ANYTIME YOU NEED TO CREATE A SINGLE VARIABLE OF A POINTER TYPE, JUST `new` TO BOTH ALLOCATE AND CONSTRUCT THE VARIABLE

YOU CAN USE THIS WITH SIMPLE TYPES (INT/FLOAT ETC) AS WELL AS WITH OBJECTS

YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

REFRESHER

# YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

```
ComplexNumber * cDynamic = new ComplexNumber(10,15);
cout << "Printing out dynamically allocated object" << endl;
cDynamic->print();
delete cDynamic;
```

DYNAMICALLY ALLOCATE + CONSTRUCT AN OBJECT!

# YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

## DYNAMICALLY ALLOCATE + CONSTRUCT AN OBJECT!

```
ComplexNumber * cDynamic = new ComplexNumber(10,15);
 cout << "Printing out dynamically allocated object" << endl;
 cDynamic->print();
 delete cDynamic;
```

```
ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
 {
   cout << "Inside the 2-argument constructor" << endl;
 }
```

IT WILL CALL THE CONSTRUCTOR WITH THE CORRESPONDING ARGUMENTS

# YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

```
ComplexNumber * cDynamic = new ComplexNumber(10,15);
 cout << "Printing out dynamically allocated object" << endl;
 cDynamic->print();
 delete cDynamic;
```

DYNAMICALLY ALLOCATE + CONSTRUCT AN OBJECT!

```
ComplexNumber(double c, double r) : realPart(r) , complexPart(c)
 {
   cout << "Inside the 2-argument constructor" << endl;
 }
```

IT WILL CALL THE CONSTRUCTOR WITH THE CORRESPONDING ARGUMENTS

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
 Dynamically assigned float has value = 23.3
 No arg-constructor called
 Inside the 2-argument constructor
 Printing out dynamically allocated object
```

AND THIS IS THE OUTPUT FROM RUNNING THE CODE

# YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

```
ComplexNumber * cDynamic = new ComplexNumber(10,15);
 cout << "Printing out dynamically allocated object" << endl;
 cDynamic->print();
 delete cDynamic;
```

REMEMBER TO USE THE -> OPERATOR FOR THIS VARIABLE!

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
Dynamically assigned float has value = 23.3
No arg-constructor called
Inside the 2-argument constructor
Printing out dynamically allocated object
real = 15 complex = 10
Inside the destructor: realPart = 15 complexPart = 10
```

AND THIS IS THE OUTPUT FROM RUNNING THE CODE

# RULE #2: USE `new/delete` FOR SINGLE VARIABLES OF ALL TYPES.

ANYTIME YOU NEED TO CREATE A SINGLE VARIABLE OF A POINTER TYPE, JUST `new` TO BOTH ALLOCATE AND CONSTRUCT THE VARIABLE

YOU CAN USE THIS WITH SIMPLE TYPES (INT/FLOAT ETC) AS WELL AS WITH OBJECTS

YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

ANYTHING YOU ALLOCATE+CONSTRUCT USING `new`, CLEAN UP USING `delete`

REFRESHER

# RULE #2: USE `new/delete` FOR SINGLE VARIABLES OF ALL TYPES.

ANYTHING YOU ALLOCATE+CONSTRUCT USING `new`, CLEAN UP USING `delete`

`new` WILL FIRST ALLOCATE MEMORY AND THEN CALL THE CONSTRUCTOR FOR YOUR VARIABLE

`delete` WILL FIRST CALL THE DESTRUCTOR, AND THEN DEALLOCATE MEMORY

REFRESHER

# ANYTHING YOU ALLOCATE+CONSTRUCT USING `new`, CLEAN UP USING `delete`

```
ComplexNumber * cDynamic = new ComplexNumber(10,15);
cout << "Printing out dynamically allocated object" << endl;
cDynamic->print();
delete cDynamic;
```

ALLOCATED + FREED USING NEW...

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
Dynamically assigned float has value = 23.3
No arg-constructor called
Inside the 2-argument constructor
Printing out dynamically allocated object
real = 15 complex = 10
Inside the destructor: realPart = 15 complexPart = 10
```

AND THIS IS THE OUTPUT FROM RUNNING THE CODE

# YOU CAN ALSO PASS IN ARGUMENTS TO THE CONSTRUCTOR

```
ComplexNumber * cDynamic = new ComplexNumber(10,15);
 cout << "Printing out dynamically allocated object" << endl;
 cDynamic->print();
 delete cDynamic;
```

ALLOCATED + FREED USING NEW...

SHOULD BE CLEANED UP USING DELETE

```
Vitthals-MacBook-Pro:~ vitthalsrinivasan$ ./a.out
Dynamically assigned float has value = 23.3
No arg-constructor called
Inside the 2-argument constructor
Printing out dynamically allocated object
real = 15 complex = 10
Inside the destructor: realPart = 15 complexPart = 10
```

AND THIS IS THE OUTPUT FROM RUNNING THE CODE