EXAMPLE 11:

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX `new/delete` AND `new[]/delete[]`

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX `new`/`delete` AND `new[]`/`delete[]`

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

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

REFRESHER

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX `new`/`delete` AND `new[]`/`delete[]`

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

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

REFRESHER

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX `new`/`delete` AND `new[]`/`delete[]`

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

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

`new[]` WILL FIRST ALLOCATE MEMORY FOR THE ARRAY, AND THEN CALL THE DEFAULT CONSTRUCTOR FOR EACH ARRAY ELEMENT. IT WILL TRACK ARRAY LENGTH TOO

`delete[]` WILL FIRST CALL THE DESTRUCTOR FOR EACH ARRAY ELEMENT, AND THEN DEALLOCATE MEMORY

REFRESHER

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX `new`/`delete` AND `new[]`/`delete[]`

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

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

REFRESHER

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX new/delete AND new[]/delete[]

```
ComplexNumber * cDynamic = new ComplexNumber[10];
  for(int i = 0;i<10;i++)
  {
    cout << "Printing out dynamically allocated object" << i << endl;
    cDynamic[i].print();
  }
  delete cDynamic;
```

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX new/delete AND new[]/delete[]

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete cDynamic;
```

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

BUT USE delete TO FREE IT - NOT delete[]

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX new/delete AND new[]/delete[]

```
ComplexNumber * cDynamic = new ComplexNumber[10];
 for(int i = 0;i<10;i++)
 {
   cout << "Printing out dynamically allocated object" << i << endl;
   cDynamic[i].print();
 }
 delete cDynamic;
```

USE new[] TO CREATE AN ARRAY OF 10 OBJECTS

BUT USE delete TO FREE IT - NOT delete[]

LET'S RUN THIS CODE AND SEE WHAT HAPPENS...

RULE #4: COROLLARY OF RULES 2 AND 3: NEVER MIX `new`/`delete` AND `new[]`/`delete[]`

USE `new[]` TO CREATE AN ARRAY OF 10 OBJECTS

BUT USE `delete` TO FREE IT - NOT `delete[]`

LET'S RUN THIS CODE AND SEE WHAT HAPPENS...

```
a.out(8540,0x7fff7684d000) malloc: *** error for object 0x7feda8403248: pointer being freed was not allocated
*** set a breakpoint in malloc_error_break to debug
Abort trap: 6
```

QED.