Students:
Section 6.26 is a part of 1 assignment: **CSC108 CH06.10-6.29 P6B**

Please browse to this assignment through BlackboardLearn so zyBooks knows where to send your activity. Learn more

Includes: PA
Due: 05/06/2025, 11:59 PM EDT

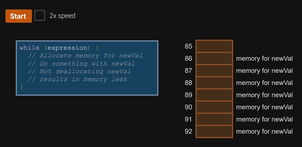
6.26 Memory leaks

Memory leak

A ***memory leak*** occurs when a program that allocates memory loses the ability to access the allocated memory, typically due to failure to properly destroy/free dynamically allocated memory. A program's leaking memory becomes unusable, much like a water pipe might have water leaking out and becoming unusable. A memory leak may cause a program to occupy more and more memory as the program runs, which slows program runtime. Even worse, a memory leak can cause the program to fail if memory becomes completely full and the program is unable to allocate additional memory.

A common error is failing to free allocated memory that is no longer used, resulting in a memory leak. Many programs that are commonly left running for long periods, like web browsers, suffer from known memory leaks — a web search for "<your-favorite-browser> memory leak" will likely result in numerous hits.

PARTICIPATION ACTIVITY 6.26.1: Memory leak can use up all available memory.



Captions

Feedback?

Garbage collection

*Some programming languages, such as Java, use a mechanism called **garbage collection** wherein a program's executable includes automatic behavior that at various intervals finds all unreachable allocated memory locations (e.g., by comparing all reachable memory with all previously-allocated memory), and automatically frees such unreachable memory. Some non-standard C++ implementations also include garbage collection. Garbage collection can reduce the impact of memory leaks at the expense of runtime overhead. Computer scientists debate whether new programmers should learn to explicitly free memory versus letting garbage collection do the work.*

PARTICIPATION ACTIVITY 6.26.2: Memory leaks.

How to use this tool

Garbage collection **Memory leak** **Unusable memory**

	Memory locations that have been dynamically allocated but can no longer be used by a program.
	Occurs when a program allocates memory but loses the ability to access the allocated memory.
	Automatic process of finding and freeing unreachable allocated memory locations.

Reset

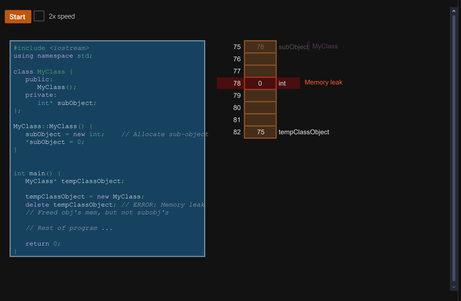
Feedback?

Memory not freed in a destructor

Destructors are needed when destroying an object involves more work than simply freeing the object's memory. Such a need commonly arises when an object's data member, referred to as a sub-object, has allocated additional memory. Freeing the object's memory without also freeing the sub-object's memory results in a problem where the sub-object's memory is still allocated, but inaccessible, and thus can't be used again by the program.

The program in the animation below is very simple to focus on how memory leaks occur with sub-objects. The class's sub-object is just an integer pointer but typically would be a pointer to a more complex type. Likewise, the object is created and then immediately destroyed, but typically something would have been done with the object.

PARTICIPATION ACTIVITY 6.26.3: Lack of destructor yields memory leak.



Captions

Feedback?

PARTICIPATION ACTIVITY 6.26.4: Memory not freed in a destructor.

1) In the above animation, which object's memory is not freed?
   - MyClass
   - tempClassObject
   - subObject

2) Does a memory leak remain when the above program terminates?
   - Yes
   - No

3) What line must exist in `MyClass`'s destructor to free all memory allocated by a `MyClass` object?
   - `delete subObject;`
   - `delete tempClassObject;`
   - `delete MyClass;`

Feedback?

PARTICIPATION ACTIVITY 6.26.5: Which results in a memory leak?

Which scenario results in a memory leak?

1)
```
int main() {
   MyClass* ptrOne = new MyClass;
   MyClass* ptrTwo = new MyClass;

   ptrOne = ptrTwo;
   return 0;
}
```
   - Memory leak
   - No memory leak

2)
```
int main() {
   MyClass* ptrOne = new MyClass;
   MyClass* ptrTwo = new MyClass;
   MyClass* ptrThree;

   ptrThree = ptrOne;
   ptrOne = ptrTwo;
   return 0;
}
```
   - Memory leak
   - No memory leak

3)
```
class MyClass {
   public:
      MyClass() {
         subObject = new int;
         *subObject = 0;
      }

      ~MyClass() {
         delete subObject;
      }

   private:
      int* subObject;
};

int main() {
   MyClass* ptrOne = new MyClass;
   MyClass* ptrTwo = new MyClass;
   ...

   delete ptrOne;
   ptrOne = ptrTwo;
   return 0;
}
```
   - Memory leak
   - No memory leak

Feedback?

How was this section? | Provide section feedback

Activity summary for assignment: CSC108 CH06.10-6.29 P6B
Due: 05/06/2025, 11:59 PM EDT

0 / 91 points

Please browse to this assignment through BlackboardLearn so zyBooks knows where to send your activity. Learn more

Completion details