CSC 211: Computer Programming

Dynamic Memory Allocation, Destructors

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Summer 2025

Course Evaluation

- Please take a few moments to fill out the course evaluation survey below:
- https://uri.campuslabs.com/eval-home/
  - On Piazza
  - Section 3000

Dynamic Memory Allocation

The `new` and `delete` operators

- Used to create and destroy variables, objects, or arrays while the program is running
- Memory allocated with the `new` operator does **NOT** use the **call stack**
  - new allocations go into the **heap** (area of memory reserved for dynamic memory allocation)
- Programmer **must** destroy all variables, objects, and arrays created dynamically
  - using the `delete` operator

Heap vs Stack

- Dynamic (heap) memory
  - allocated during run time
  - exact sizes or amounts don't need to be known
  - must use pointers
  - alternative to local stack memory
- Static (stack) memory
  - exact size and type of memory must be known at compile time.
  - local variables are allocated automatically when a function is called and they are deallocated automatically when the function exits.

When do we need dynamic memory?

- **When you need a lot of memory.**
  - Typical stack size is 1 MB, so anything bigger than 50-100KB should better be dynamically allocated, or you're risking crash.
- **When the memory must live after the function returns.**
  - Stack memory gets destroyed when function ends, dynamic memory is freed when you want.
- **Size that is unknown at runtime**
  - When you're building a structure (like array, or graph) that dynamically changes or is too hard to precalculate.
- **Allocate storage space while the program is running**
  - We cannot create new variable names "on the fly"

Then why does this work?

- There is a GCC extension to the standard that makes this work
- Not part of the standard C++ specification, but it is supported by some compilers as an extension from the C99 standard of the C language.

```
int n = 0;
int i = 0;

std::cout << "Enter size: ";
std::cin >> n;
int myarray[n];

for (i=0; i<n; i++)
{
  myarray[i] = i;
}
```



```
#include <iostream>

int main( ) {
    int *p1, *p2;

    p1 = new int;
    *p1 = 10;
    p2 = p1;
    *p2 = 20;
    p1 = new int;
    *p1 = 30;

    std::cout << *p1 << ' ' << *p2 << '\n';

    delete p1;
    delete p2;

    return 0;
}
```

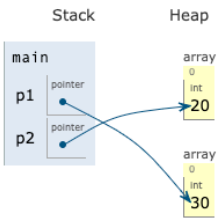
Tracing the code

C++ (gcc 4.8, C++11)
EXPERIMENTAL! known limitations

```
1  #include <iostream>
2
3  int main( ) {
4      int *p1, *p2;
5
6      p1 = new int;
7      *p1 = 10;
8      p2 = p1;
9      *p2 = 20;
10     p1 = new int;
11     *p1 = 30;
12
13     std::cout << *p1 << ' ' << *p2 << '\n';
14
15     delete p1;
16     delete p2;
17
18     return 0;
19 }
```

Print output (drag lower right corner to resize)

```
30 20
```



http://pythontutor.com/cpp.html#mode=edit

Syntax for new and delete

```
#include "date.h"
#include <iostream>

int main( ) {
    // creating a single variable
    int *p = new int;
    *p = 5;

    // creating an array
    int *array = new int[20];
    for (int i = 0 ; i < 20 ; i ++) {
        array[i] = 0;
    }

    // creating an object
    Date *today = new Date(11, 18, 2019);
    today->print();

    // delete all allocated objects
    delete p;
    delete [] array;
    delete today;

    return 0;
}
```

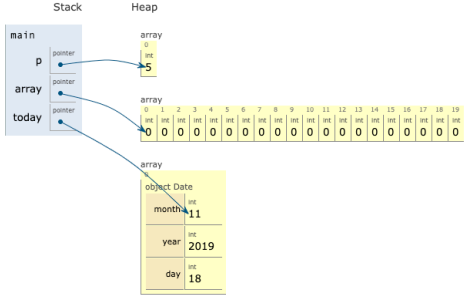

Tracing the code

C++ (gcc 4.8, C++11)
EXPERIMENTAL! known limitations

```
31 int main( ) {
32     // creating a single variable
33     int *p = new int;
34     *p = 5;
35
36     // creating an array
37     int *array = new int[20];
38     for (int i = 0 ; i < 20 ; i ++) {
39         array[i] = 0;
40     }
41
42     // creating an object
43     Date *today = new Date(11, 18, 2019);
44     (*today).print();
45
46     // delete all allocated objects
47     delete p;
48     delete [] array;
49     delete today;
50
51     return 0;
52 }
```

Print output (drag lower right corner to resize)

```
11-18-2019
```



http://pythontutor.com/cpp.html#mode=edit

Array Resizing

```
int size = 5;

int * list = new int[size];

for(int i =0; i < 5; i++){
    list[i] = i;
}

/// need to add more space later on

int * temp = new int[size + 5];

 for (int i = 0; i < size; i++){
    temp[i] = list[i];
 }

  delete [] list;  // this deletes the array pointed to by "list"

   list = temp;
```

https://pythontutor.com

Pointers and objects

- Data members and methods of an object can be accessed by dereferencing a pointer

```
Date *today = new Date(11, 18, 2019);
(*today).print();
```

- Or … can use the **–> operator**

```
Date *today = new Date(11, 18, 2019);
today->print();
```


malloc() / free()

malloc()

- The function malloc() is used to allocate the requested size of bytes and it returns a pointer to the first byte of allocated memory. It returns null pointer, if fails.
- Here is the syntax of malloc() in C++ language,

```
pointer_name = (cast-type*) malloc(size);
```

- **pointer_name** – Any name given to the pointer.
- **cast-type** – The datatype in which you want to cast the allocated memory by malloc().
- **size** – Size of allocated memory in bytes.

```
float* ptr = (float *) malloc (10 * sizeof(float));
```


free()

- The function free() is used to deallocate the allocated memory by malloc(). It does not change the value of the pointer which means it still points to the same memory location.
- Here is the syntax of free()

```
void free(void *pointer_name);
```

- **pointer_name** – Any name given to the pointer.

```
free(ptr);
```

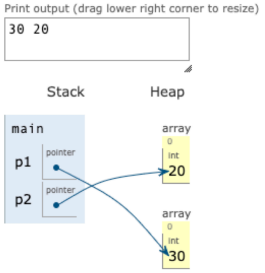
Malloc/Free in use

```
int main() {
   int n = 4, i, *p, mystery = 0;
   p = (int*) malloc(n * sizeof(int));
   if(p == NULL) {
      std::cout << ("Error! memory not allocated.");
   }
   std::cout << ("Enter elements of array : ");
   for(i = 0; i < n; ++i) {
      std::cin >> (*p);
      mystery += (*p);
      p++;
   }

   std::cout << (sum);

   free(p);

   return 0;
}
```


Dangling Pointers

Dangling Pointers

- Pointer remains allocated but pointed to memory is deallocated.
- Remember: The pointer is statically allocated, but the memory it's pointing to is dynamically allocated.




```
int* ptr = new int;
delete ptr;

// Memory is deallocated, but `ptr` still holds the
address.

// Accessing it is undefined behavior:
std::cout << *ptr; // Undefined behavior!

// Safer approach:
ptr = nullptr;
```

Dangling Pointer!

Memory Leaks

Memory Leak

- Pointer is deallocated but pointed to memory is remains allocated.
- That memory is no longer in use by the program. So that memory location is reserved for no reason.
- Opposite of a dangling pointer

Memory Leak

```
void my_func() {
   int *data = new int;
   *data = 50;
}
```

👎

```
void my_func() {
   int *data = new int;
   *data = 50;
   delete data;
}
```

👍

Destructors

Destructor

- Special `method` automatically called when objects are destroyed
  - it is used to delete any memory created **dynamically**
- Objects are destroyed when …
  - … they exist in the stack and go out of scope
  - … they exist in the heap and the delete operator is used
- A destructor …
  - … is a member function (usually `public`)
  - … must have the same name as its class preceded by a ~
  - … is automatically called when an object is destroyed
  - … does not have a return type (not even `void`)
  - … takes no arguments

Destructor Syntax

```
//Syntax for defining the destructor within the class
~ <classname>()
{
//body
}


//Syntax for defining the destructor outside the class
<classname>::~<classname>()
{
//body
}
```


Destructor Syntax

```
class Test
{
    public:
        Test()
        {
            std::cout<<"\n Constructor executed";
        }

        ~Test()
        {
            std::cout<<"\n Destructor executed";
        }
};

int main(){

    Test t,t1,t2,t3;
    return 0;
}
```


Destructor Syntax

```
class Test
{
    public:
        Test()
        {
            std::cout<<"\n Constructor executed";
        }

        ~Test()
        {
            std::cout<<"\n Destructor executed";
        }
};

int main(){

    Test t,t1,t2,t3;
    return 0;
}
```

```
 Constructor executed
 Constructor executed
 Constructor executed
 Constructor executed
 Destructor executed
 Destructor executed
 Destructor executed
 Destructor executed%
michaelconti@Michaels-MacBook-Pro-2 Desktop %
```

Try it

- Write a program that stores the GPA of n number of students using dynamic memory.