COMP 1633: Intro to CS II

Dynamic Allocation and Midterm Review

Charlotte Curtis

March 4, 2024

Where we left off

- Pointers and arrays
- Pointers and structures
- Pointers and functions
- `typedef`
- Preview of dynamic memory allocation

> *Textbook Sections 9.1, 9.2*

```
Time t;
Time *pt = &t;

t.hour = 5;
t.minute = 0;
cout << pt->hour << ':'
     << pt->minute << endl;
```

Today's topics

- Dynamic memory allocation
- Midterm review exercise

> *Textbook Sections 9.2, 6.1*

The heap and the stack

- There are two accessible areas of memory for a program:
  - The **stack** is used for local variables and function calls
  - The **heap** (or "freestore") is used for dynamic memory allocation

The `new` operator

To create a variable on the heap, use the `new` operator:

```
int x = 0; // x is a named memory location on the stack
int *ptr; // memory for pointer is on the stack
ptr = new int; // what it points at is on the heap
```

- By using `new`, we tell C++ that we want the memory to be allocated on the **heap**
- The **only way** to access the value at `ptr` is through the pointer
- What if we do the following?

  ```
  ptr = &x;
  ```

- If you lose the address of the pointer, your integer is lost and gone forever!

`new` structures

- Creating an `int` on the heap is a bit silly, they don't take up much space anyway
- More useful for a `struct`:

```
Applicant *a = new Applicant; // Allocates all 19 fields on the heap
```

- Recall the pointer + struct syntax:

```
strcpy(a->name, "Aaron Grimm");
cout << a->name << endl;
```

Every `new` needs a `delete`

- After allocating space on the heap (for an `Applicant` or an `int` or anything else), you should **free the memory** using `delete` when you're done with it
- This prevents memory leaks
- Syntax:

  ```
  delete a;
  ```

  where `delete` is an **operator** and the operand is the **pointer variable name**

> *Caution: this recycles the **memory**, but does not remove the **pointer!***
> *Good idea to reset the pointer to `NULL` after a `delete`*

Summary of `new` and `delete`

`new`	`delete`
Allocates memory on the heap	Returns memory to the heap
Returns a pointer to the allocated memory	Does not modify the pointer address

Risks:

- Memory leaks - forgetting to `delete` a pointer
- Dangling pointers - `delete`ing a pointer and then trying to use it
- Double `delete` - `delete`ing a pointer twice
- `delete`ing a pointer that was not created with `new`

Allocating variable sized arrays

- To create a **variable sized array**, we need to use `new`:

```cpp
int n;
cin >> n;
int *arr = new int[n];
```

- This allocates contiguous memory on the heap for `n` integers
- We can then use the array the way we normally would:

```cpp
arr[0] = 5;
a_func_that_uses_an_array(arr, n);
```

- `delete`ing an array needs a bit of extra syntax:

```cpp
delete [] arr;
```

Static vs dynamically allocated arrays

Static	Dynamic
Size must be known at compile time	Size can be variable
Memory allocated on the **stack**	Memory allocated on the **heap**
Memory freed automatically when variable goes out of scope	Must be manually `delete`d when you're done with it
Limited by stack size	Limited by system memory
Contiguous memory	Contiguous memory

Midterm review exercise

Pub trivia style! Answers are now posted.

- Groups of 3-4
- I'll read questions out loud, you have 2 minutes per question to discuss and write down your answers.
- **Do not shout out answers** - write them down and we'll peer mark at the end.

Q5:

```
int x = 5;
int *p1;
int *p2 = &x;
```

Q9:

```
int nums[8], n;
cin >> n;
for (int i = 0; i < n; i++) {
    cin >> nums[i];
}
```

Coming up next

- Tomorrow's lab: drop in help/study session
- **Midterm** 🎉 on Wednesday
- Thursday's lab: Dynamic allocation and valgrind