Dynamic Allocation of Memory

Lecture 5
Section 9.8

Robb T. Koether

Hampden-Sydney College

Wed, Jan 24, 2018

Outline

The `Vectr` Class

- We would like to design a class of objects that would be like arrays, but whose size could be declared at run time, not compile time.
- We will call this class `Vectr`. (There is already a `vector` class.)

Arrays vs. `Vectrs`

Arrays

```
const int MAX_SIZE = 20;
int size = 10;
int a[MAX_SIZE];          // Legal
int b[size];              // Illegal
a[5] = 123;               // Legal
a = b;                    // Illegal
cout << a[5] << endl;     // Legal
cout << a << endl;        // Legal, but not advised. Why?
```

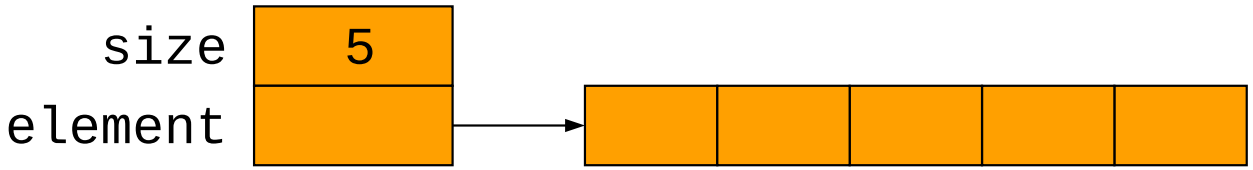
Arrays vs. Vectrs

Vectrs

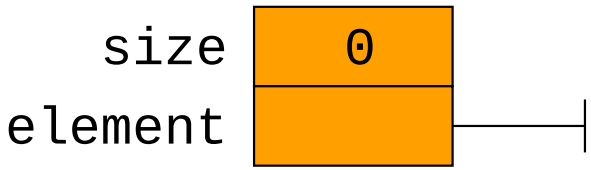
```
const int MAX_SIZE = 20;
int size = 10;
Vectr v(MAX_SIZE);      // Legal
Vectr w(size);          // Legal
v[5] = 123;             // Legal
v = w;                  // Legal
cout << v[5] << endl;   // Legal
cout << v << endl;      // Legal
```

`Vectr` Objects

- Non-empty `Vectr`



- Empty `Vectr`



Outline

Allocation of Memory: C Style

The `malloc()` Prototype

```
void* malloc(int number-of-bytes);
```

- The library function `malloc()` allocates a specified number of bytes of memory and returns a pointer to it.
- Include the header file `<cstdlib>`.

Allocation of Memory: C Style

`malloc()` Usage

```
int* pi = (int*)malloc(sizeof(int));
Point* ppt = (Point*)malloc(sizeof(Point));
```

- `malloc()` returns a pointer to the first byte of the allocated memory block.
- The returned pointer is a pointer to **`void`**.
- The returned pointer must be *cast* to the proper type.

Outline

Deallocation of Memory: C Style

The `free()` Prototype

```
void free(void* p);
```

- The library function `free()` deallocates memory.
- Include the header file `<cstdlib>`.
- The pointer *must* contain an address that was previously returned by `malloc()`.

Deallocation of Memory: C Style

The `free()` Usage

```
int* p = (int*)malloc(sizeof(int));
*p = 123;       // Do stuff with p
    ⋮
free(p);
```

- If the programmer does not call `free()`, then memory allocated by `malloc()` is automatically freed when the program exits.

Dynamic Memory for Arrays: C Style

Array Allocation with `malloc()` and `free()`

```
int size = 20;
int* p = (int*)malloc(size*sizeof(int));
p[5] = 123;      // Do stuff with p
    ⋮
free(p);
```

- `malloc()` can be used to allocate memory for an array.
- Then `free()` will deallocate the memory.
- The computer remembers the size of the array.

Example

Example (Example)

- `DynamicCArray.cpp`.

Outline

The `calloc()` Function

The `calloc()` Prototype

```
void* calloc(int num-of-objects, int size-of-object);
```

- The library function `calloc()` allocates memory for a specified number of objects each of a specified size and returns a pointer to it.
- Include the header file `<cstdlib>`.

The `calloc()` Function

`calloc()` Usage

```
int* pi = (int*)calloc(1, sizeof(int));
Point* ppt_arr = (Point*)calloc(50, sizeof(Point));
```

- `calloc()` returns a pointer to the first byte of the allocated memory block.
- The returned pointer is a pointer to **`void`**.
- The returned pointer must be *cast* to the proper type.

The `realloc()` Function

The `realloc()` Prototype

```
void* realloc(void* p, int num-of-bytes);
```

- The library function `realloc()` will allocate a new block of memory containing the specified number of bytes.
- The contents of the “old” memory will be copied to the “new” memory (as much as fits).
- Include the header file `<cstdlib>`.

The `realloc()` Function

`realloc()` Usage

```
int* p = (int*)malloc(100*sizeof(int));
for (int i = 0; i < 100; i++)
    p[i] = 10*i;
p = (int*)realloc(p, 200*sizeof(int));
```

- The contents `0`, `10`, `20`, `...`, `990` will be copied to the new memory.

Outline

Outline

Allocation of Memory: C++ Style

The `new` Operator

```
Type* p = new Type;          // For single object
Type* p = new Type[size];    // For an array
```

- C++ introduced the **`new`** operator to replace `malloc()`.
- It can allocate memory for a single object.
- And it can allocate memory for an array of objects.

Outline

Deallocation of Memory: C++ Style

The `delete` Operator

```
delete p;      // Delete single object
delete [] p;   // Delete an array
```

- The **`delete`** operator will delete memory that was allocated by the **`new`** operator.
- **`delete`** can deallocate memory for a single object.
- And it can deallocate memory for an array of objects.
- The pointer *must* contain an address that was previously returned by **`new`**.

Dynamic Memory for Arrays: C++ Style

Array Allocation with **new** and **delete**

```
int size = 20;
int* p = new int[size];
p[5] = 123;      // Do stuff with p
    ⋮
delete [] p;
```

Example

Example (Example)

- `DynamicC++Array.cpp`.

Outline

Memory Leaks

Definition (Memory Leak)

A memory leak occurs when all pointers to a block of allocated memory have been lost.

- Leaked memory cannot be accessed or reallocated; it is useless.
- Excessive memory leaks may cause the program to run out of usable memory and crash.
- Memory leaks should *always* be avoided.

Outline

Dangling Pointers

Definition (Dangling Pointer)

A dangling pointer is a non-null pointer that points to unallocated memory.

- Dereferencing a dangling pointer may cause the program to crash.
- We do not necessarily avoid dangling pointers, but we must be careful.

Avoiding Dangling Pointers

- It impossible to test a non-null pointer to see whether it is dangling.
- Always set pointers to `NULL` if they do not point to allocated memory.
- Then compare them to `NULL` to see whether they point to allocated memory.

Outline

Assignment

Assignment

- Read Section 9.8.