CS101: An Introduction to Programming through C++ 2026

Lecture 18: Heap

Instructor: Ashutosh Gupta

IITB India

Compile date: April 14, 2026

Based on material developed by Prof. Abhiram Ranade and Prof. Manoj Prabhakaran

Storing a collection of things

Global/local arrays store a fixed-sized collection of things in a contiguous memory segment.

- We may not know the size of the collection at the start.
- The memory requirement may change over time. Therefore, inefficient use of memory.
- Large swath of contiguous memory may not be available.

Solution: Dynamically allocated memory

Topic 18.1

Dynamic memory allocation

Dynamic allocation of memory

- Syntax:

```
type* pointer = new type;
```

- Example:

```
int* p = new int;                    //creates a new object
*p = 7;                              //Write a value to the object
cout << p << ", "<< *p << endl;//Read the value of the object
```

- A `new` expression creates a new "object" in the memory.
- The object has no name.
- `new` returns a pointer to the object, which can be accessed via the pointer.

Exercise 18.1 (Edit new.cpp)

Create another float object. Compare the addresses.

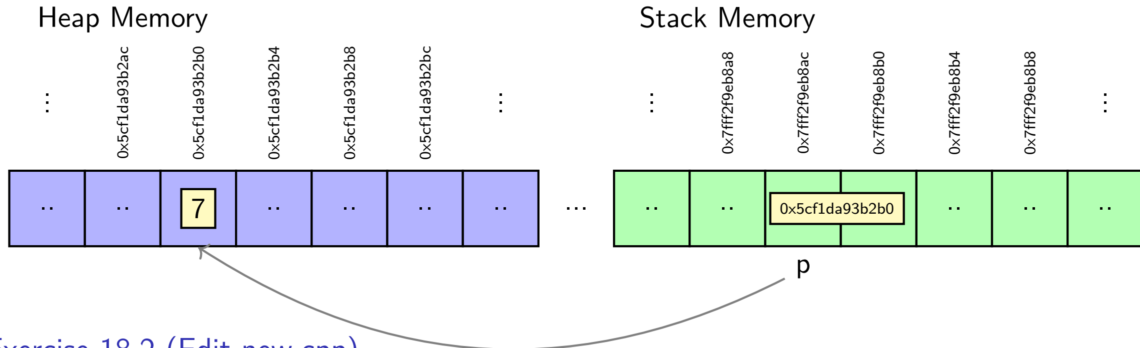
Parts of memory

For a program, the memory is divided into four segments.

- Memory for program code
- Memory for global/static variables
- Stack: stores local variables
- Heap: stores dynamically allocated memory

Example: Parts of memory

```
int main() {
 int* p = new int;
 *p = 7;
}
```

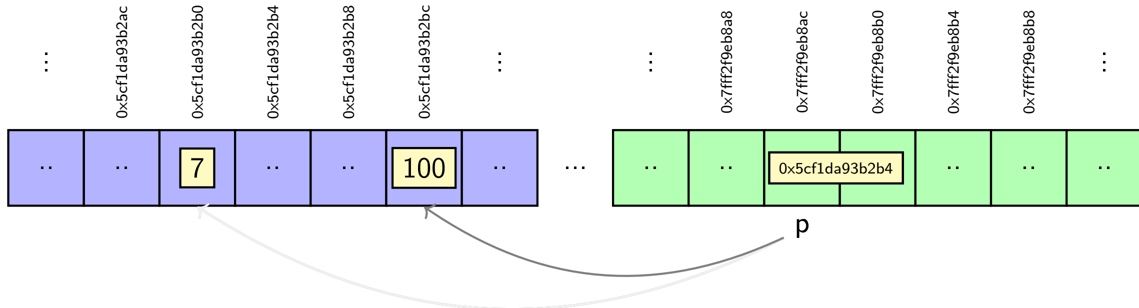


Exercise 18.2 (Edit new.cpp)

Check the memory location of a global variable. Is it before stack memory or heap memory?

A new phenomenon: memory leak, be careful!

```
main() {
 int* p = new int;
 *p = 7;
 p = new int;   // Memory leak! We lost access to the previous object!
 *p = 100;      // Object containing 7 still exists, but is inaccessible
}
```



Exercise 18.3 (Edit .vscode/tasks.json)

Add option `-fsanitize=address` in the compilation of leak.cpp?

The one who creates must also destroy!

- Syntax:

```
delete pointer;
```

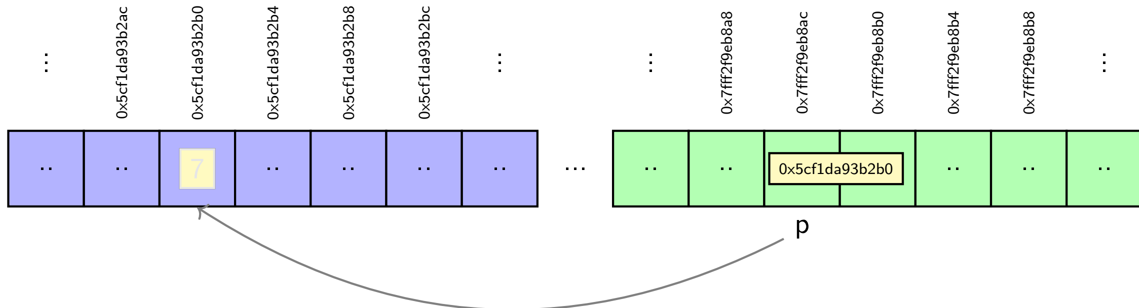
- Example:

```
int* p = new int;
*p = 7;
delete p; //Object is deleted from the heap memory!
```

- Deletes the object from the memory that is pointed to by the pointer
- The memory address continues to exist, and the pointer continues to point at the address
- Accessing the object after the deletion is an undefined behavior.

Example: delete

```
int main() {
 int* p = new int;
 *p = 7;
 delete p;
}
```



Exercise 18.4 (Edit delete.cpp)

What happens if we delete the same memory multiple times or an object in stack memory?

Topic 18.2

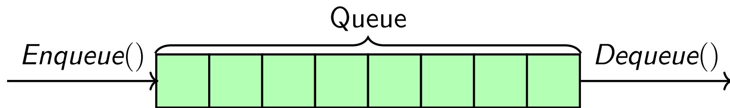
An application of pointers:Queue

Queue

Definition 18.1

Queue is a container where elements are added and deleted according to the first-in-first-out (FIFO) order.

- Addition is called enqueue
- Deleting is called dequeue



Example 18.1

- Entry into an airport
- Connection requests in a web server

Interface of the queue

The queue supports four main interface functions.

- `queue q` : allocates a new queue q.
- `q.enqueue(e)` : Adds the given element e to the end of the queue.
- `q.dequeue()` : Removes the first element from the queue.

Queue using allocated memory!

We may not know the maximum size of the queue.

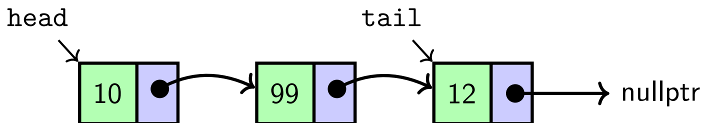
Using dynamically allocated memory, we can implement queue.

We use a linked list for this purpose.

Linked lists

Definition 18.2

A linked list consists of nodes (small pieces of memory) with two fields: data and a next pointer. The nodes form a chain via the next pointer. The data contains the objects that are stored on the linked list. We need two pointer, head and tail, to access the linked list.



Exercise 18.5

a. If we use a linked list for implementing a queue, which side should be the front of the queue?

b. What is the fundamental difference between an array and a linked list?

Declaring a linked list

- Declaring a node

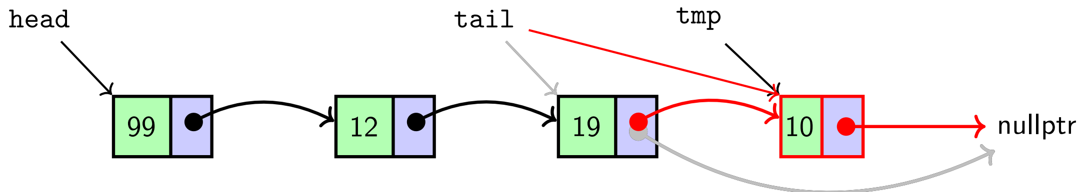
```
struct node {
  int data;
  node* next = nullptr;
};
```

- Declaring a queue to access the nodes

```
struct queue {
  node* head = nullptr;
  node* tail = nullptr;
  void enqueue(int v);
  bool dequeue(int* v);//Why are we passing input parameter?
  print();    // Prints all the elements of the queue
};
```

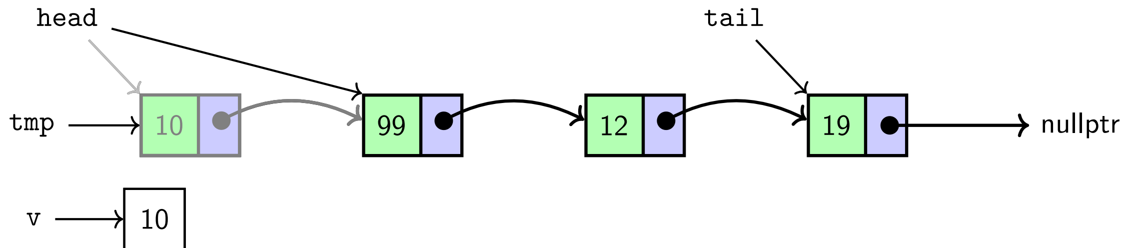
Enqueue in linked lists

```
void enqueue(int v) {
 if(!tail){//If the queue is empty, update both head and tail
     tail = head = new node;
   }else {
    node* tmp = new node; //Create a new node
    tail->next = tmp;     //Connect the new node to the list
    tail = tail->next;    //Move the pointer
   }
 tail->data = v;
}
```



Dequeue in linked lists

```
bool dequeue(int* v) {
 if(!head) return false; //if head is null, nothing to remove
 *v = head->data;
 node* tmp = head;
 head = head->next;
 delete tmp;
 if(!head) tail=nullptr; // if head is null, tail must also be updated
 return true;
}
```



Printing all elements of the queue

```
void print() {
  for(node* p = head; p; p = p->next)
    cout << p->data << " ";
  cout << endl;
}
```

Exercise 18.6 (Edit queue.cpp)

a. What will happen if we remove condition p in the for loop?

b. Can a piece of memory be allocated at address 0x0000000000000000?

c. In the queue, add an interface for reading the front element.

Hidden memory leak

```
void queue_leak_demo() {
  queue Q;
  for(int i=0; i < 10000000; i++)
    Q.enqueue(1);
} // Memory Leak!
```

When Q goes out of scope, its "contents" allocated using new are not deleted!

Clearing memory

We need to add the following function to the queue structure to deallocate all nodes.

```
struct queue {
  .....
  void clear() {
    int v;
    while (dequeue(&v)) {}
  }
};
void queue_leak_demo() {
  queue Q;
  for(int i=0; i < 10000000; i++) Q.enqueue(1);
  Q.clear(); // Whenever Q goes out of scope, we need to call clear()
}            // Can we automate this?
```

Exercise 18.7 (Edit queue.cpp)

Comment call to clear() and observe the memory leak using the top command!

Topic 18.3

Variable-length array

Allocating a variable-length array on the heap

- Syntax:

```
type* var = new type[n];
```

- returns a pointer to the first element in an array of n elements of the requested type

- Free syntax:

```
delete[] var;
```

- Caution: Whenever we use `new` or `new[]` in our program, we need to make sure there is a matching delete or delete[]

Example: reading inputs of given length

```
int main() {
  int n; cin >> n;
  int* numbers = new int[n];//instead of relying on VLA support: int p[n];
  for(int i=0; i<n; i++) cin >> numbers[i];
  for(int i=n; i>0; i--) maxend(numbers,i);//Sorting
  for(int i=0; i<n; i++) cout << numbers[i] << " ";
  cout << endl;
  delete[] numbers; //release the allocated memory
}
```

Exercise 18.8 (Edit sorting.cpp)

Recall global arrays are always initialized. Check if the dynamically allocated arrays are initialized.

Topic 18.4

Problems

Size of queue

Exercise 18.9

Modify queue struct such that we can get size of the queue.

Exercise: reversing a linked list*

Exercise 18.10

Give an algorithm to reverse a linked list. You must use only three extra pointers.

Exercise: middle element*

Exercise 18.11

Give an algorithm to find the middle element of a linked list.

End of Lecture 18