

Last Week

- Memory, a Program's Perspective
- More on Variables
- Memory Management
- Simple Pointers, Pointer Types and Casting
- Pointers and Arrays
- Complex Pointers, Function Pointers

Overview

- A More Complex Pointer Example
- Compiling, linking, makefiles, debugging, etc.

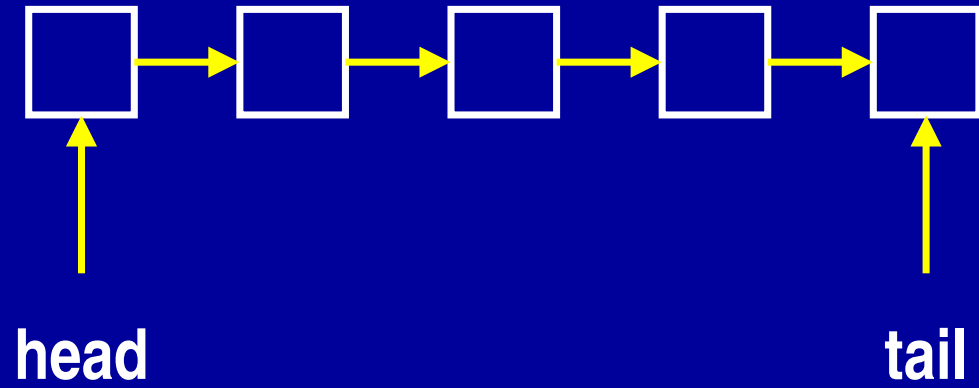
What you already know ...

- Showed that pointers are useful for
 - ★ passing parameters by reference
 - ★ using dynamically allocated memory
- Pointers are equal (sort of) to arrays
 - ★ also multi dimensional arrays
- Also pointer pointers, pointer pointer pointers, etc.

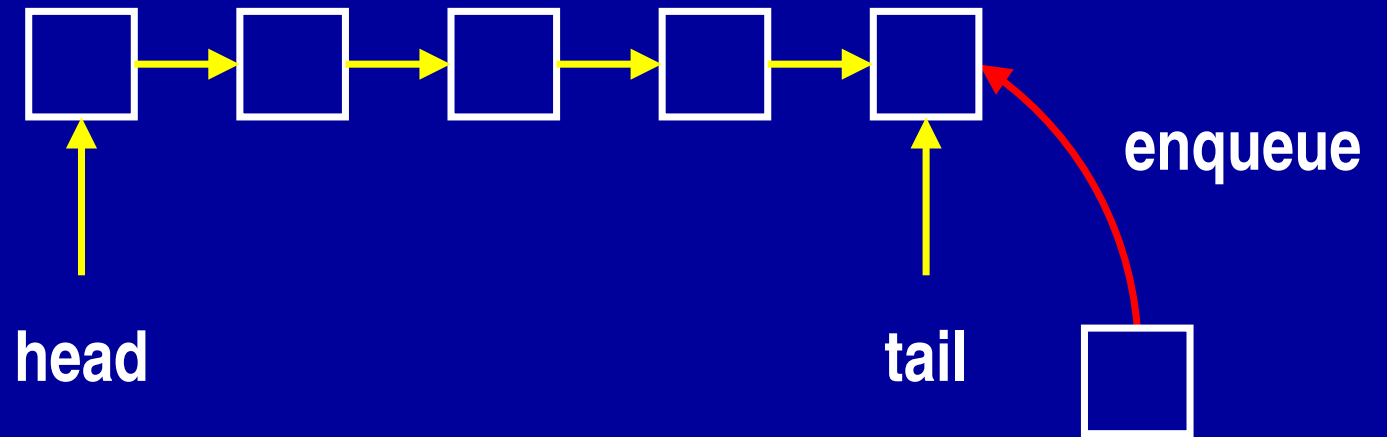
Complex Datastructures (1)

- You can use a combination of **structs**, **pointers** and **memory allocation** to create complex datastructures
- Queues, lists, trees, hashtables ...
- Will use a **queue** as an example

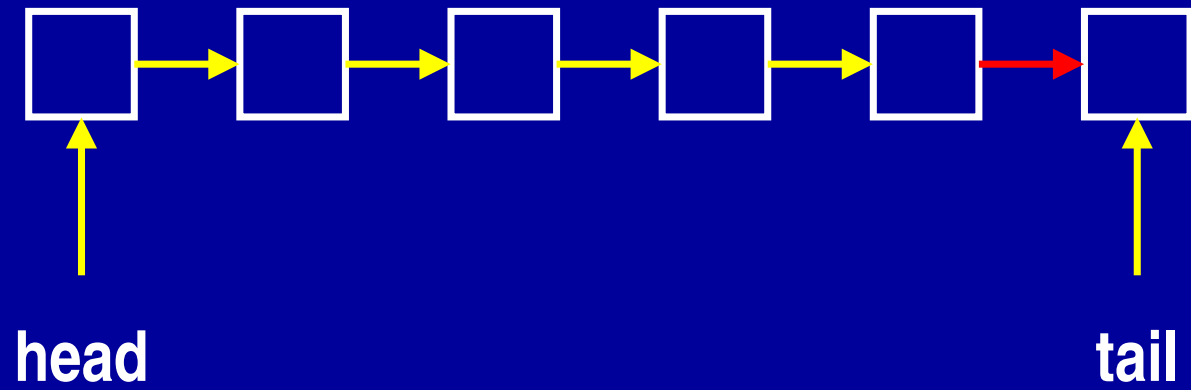
Queue (2)



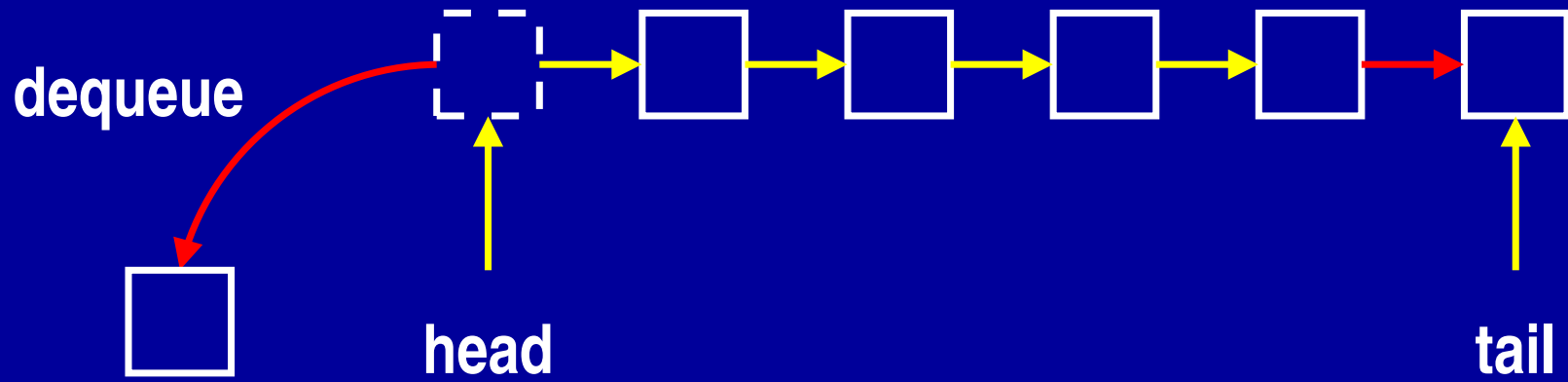
Queue (2)



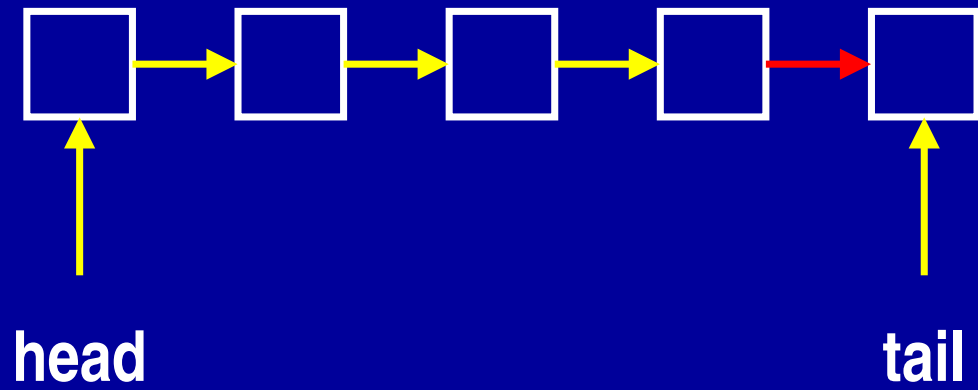
Queue (2)



Queue (2)



Queue (2)



Queue (3)

- Queue
 - ★ adds at tail (enqueue)
 - ★ removes at head (dequeue)
- Want a **generic** queue
 - ★ can contain any type of data

Queue in Java (1)

```
package utils;
public class Queue {
    private Node head;
    private Node tail;
    private class Node {
        Object data;
        Node next;
        Node(Object data) {
            this.data = data;
            next = null;
        }
    }
    /* ... */
}
```

Queue in Java (1)

```
-> package utils;
    public class Queue {
        private Node head;
        private Node tail;

        private class Node {
            Object data;
            Node next;

            Node(Object data) {
                this.data = data;
                next = null;
            }
        }

        /* ... */
    }
```

Queue in Java (1)

```
package utils;
public class Queue {
->     private Node head;
->     private Node tail;
    private class Node {
        Object data;
        Node next;
        Node(Object data) {
            this.data = data;
            next = null;
        }
    }
    /* ... */
}
```

Queue in Java (1)

```
package utils;
public class Queue {
    private Node head;
    private Node tail;
-> private class Node {
->     Object data;
->     Node next;
->
->     Node(Object data) {
->         this.data = data;
->         next = null;
->     }
-> }
->     /* ... */
}
```

Queue in Java (2)

```
package utils;
public class Queue {
    /* ... */
    public void enqueue(Object data) {
        Node node = new Node(data);
        if (tail == null) {
            // Queue is empty
            head = tail = node;
        } else {
            tail.next = node;
            tail = node;
        }
    }
    /* ... */
}
```

Queue in Java (2)

```
package utils;
public class Queue {
    /* ... */
    public void enqueue(Object data) {
->        Node node = new Node(data);
        if (tail == null) {
            // Queue is empty
            head = tail = node;
        } else {
            tail.next = node;
            tail = node;
        }
    }
    /* ... */
}
```

Queue in Java (2)

```
package utils;
public class Queue {
    /* ... */
    public void enqueue(Object data) {
        Node node = new Node(data);
        ->     if (tail == null) {
        ->         // Queue is empty
        ->         head = tail = node;
        ->     } else {
        ->         tail.next = node;
        ->         tail = node;
        ->     }
        }
    /* ... */
}
```

Queue in Java (3)

```
package utils;
public class Queue {
    /* ... */
    public Object dequeue() {
        if (head == null) {
            return null;
        } else {
            Node temp = head;
            if (head == tail) {
                head = tail = null;
            } else {
                head = head.next;
            }
            return temp.data;
        }
    }
}
```

Queue in Java (3)

```
package utils;
public class Queue {
    /* ... */
    public Object dequeue() {
->         if (head == null) {
->             return null;
        } else {
            Node temp = head;
            if (head == tail) {
                head = tail = null;
            } else {
                head = head.next;
            }
            return temp.data;
        }
    }
}
```

Queue in Java (3)

```
package utils;
public class Queue {
    /* ... */
    public Object dequeue() {
        if (head == null) {
            return null;
        } else {
            Node temp = head;
            if (head == tail) {
                head = tail = null;
            } else {
                head = head.next;
            }
            return temp.data;
        }
    }
}
```

Queue in Java (4)

```
import utils.Queue;
class Main {
    public static void main(String [] args) {
        Queue queue = new Queue();
        for (int i=0;i<100;i++) {
            queue.enqueue(new Integer(i));
        }
        for (int i=0;i<102;i++) {
            Integer integer = (Integer) queue.dequeue();
            System.out.println("Dequeued " + integer);
        }
    }
}
```

Queue in Java (4)

```
-> import utils.Queue;
class Main {
    public static void main(String [] args) {
        Queue queue = new Queue();
        for (int i=0;i<100;i++) {
            queue.enqueue(new Integer(i));
        }
        for (int i=0;i<102;i++) {
            Integer integer = (Integer) queue.dequeue();
            System.out.println("Dequeued " + integer);
        }
    }
}
```

Queue in Java (4)

```
import utils.Queue;
class Main {
    public static void main(String [] args) {
->        Queue queue = new Queue();
        for (int i=0;i<100;i++) {
            queue.enqueue(new Integer(i));
        }
        for (int i=0;i<102;i++) {
            Integer integer = (Integer) queue.dequeue();
            System.out.println("Dequeued " + integer);
        }
    }
}
```

Queue in Java (4)

```
import utils.Queue;
class Main {
    public static void main(String [] args) {
        Queue queue = new Queue();
->     for (int i=0;i<100;i++) {
->         queue.enqueue(new Integer(i));
->     }
->     for (int i=0;i<102;i++) {
->         Integer integer = (Integer) queue.dequeue();
->         System.out.println("Dequeued " + integer);
    }
}
```

Queue in C: queue.h

```
#ifndef UTILS_QUEUE
#define UTILS_QUEUE

struct queue;

typedef struct queue Queue;

Queue *createQueue();
void deleteQueue(Queue *queue, int freeData);

void enqueue(Queue *queue, void *data);
void *dequeue(Queue *queue);

#endif
```

Queue in C: queue.h

```
→ #ifndef UTILS_QUEUE
→ #define UTILS_QUEUE

    struct queue;

    typedef struct queue Queue;

    Queue *createQueue();
    void deleteQueue(Queue *queue, int freeData);

    void enqueue(Queue *queue, void *data);
    void *dequeue(Queue *queue);

→ #endif
```

Queue in C: queue.h

```
#ifndef UTILS_QUEUE
#define UTILS_QUEUE
```

```
→ struct queue;
```

```
typedef struct queue Queue;
```

```
Queue *createQueue();
```

```
void deleteQueue(Queue *queue, int freeData);
```

```
void enqueue(Queue *queue, void *data);
```

```
void *dequeue(Queue *queue);
```

```
#endif
```

Queue in C: queue.h

```
#ifndef UTILS_QUEUE
#define UTILS_QUEUE
```

```
struct queue;
```

→ `typedef struct queue Queue;`

```
Queue *createQueue();
```

```
void deleteQueue(Queue *queue, int freeData);
```

```
void enqueue(Queue *queue, void *data);
```

```
void *dequeue(Queue *queue);
```

```
#endif
```

Queue in C: queue.h

```
#ifndef UTILS_QUEUE
#define UTILS_QUEUE

struct queue;

typedef struct queue Queue;

-> Queue *createQueue();
-> void deleteQueue(Queue *queue, int freeData);

void enqueue(Queue *queue, void *data);
void *dequeue(Queue *queue);

#endif
```

Queue in C: queue.h

```
#ifndef UTILS_QUEUE
#define UTILS_QUEUE

struct queue;

typedef struct queue Queue;

Queue *createQueue();
void deleteQueue(Queue *queue, int freeData);

-> void enqueue(Queue *queue, void *data);
-> void *dequeue(Queue *queue);

#endif
```

Queue in C: queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "queue.h"

struct node {
    void *data;
    struct node *next;
};

typedef struct node Node;

struct queue {
    Node *head;
    Node *tail;
};

/* ... */
```

Queue in C: queue.c

```
-> #include <stdlib.h>
-> #include <stdio.h>
#include "queue.h"

struct node {
    void *data;
    struct node *next;
};

typedef struct node Node;

struct queue {
    Node *head;
    Node *tail;
};

/* ... */
```

Queue in C: queue.c

```
#include <stdlib.h>
#include <stdio.h>
-> #include "queue.h"

struct node {
    void *data;
    struct node *next;
};

typedef struct node Node;

struct queue {
    Node *head;
    Node *tail;
};

/* ... */
```

Queue in C: queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "queue.h"

-> struct node {
->     void *data;
->     struct node *next;
-> };

typedef struct node Node;

struct queue {
    Node *head;
    Node *tail;
};

/* ... */
```

Queue in C: queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "queue.h"

struct node {
    void *data;
    struct node *next;
};

-> typedef struct node Node;

struct queue {
    Node *head;
    Node *tail;
};

/* ... */
```

Queue in C: queue.c

```
#include <stdlib.h>
#include <stdio.h>
#include "queue.h"

struct node {
    void *data;
    struct node *next;
};

typedef struct node Node;

-> struct queue {
->     Node *head;
->     Node *tail;
-> };

/* ... */
```

Queue in C: queue.c

```
/* ... */
Queue *createQueue()
{
    Queue *queue = malloc(sizeof(Queue));
    if (queue == NULL) {
        printf("Out of memory!\n");
        exit(1);
    }
    queue->head = NULL;
    queue->tail = NULL;
    return queue;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
Queue *createQueue()
{
→   Queue *queue = malloc(sizeof(Queue));
   if (queue == NULL) {
       printf("Out of memory!\n");
       exit(1);
   }
   queue->head = NULL;
   queue->tail = NULL;
   return queue;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
Queue *createQueue()
{
    Queue *queue = malloc(sizeof(Queue));
->    if (queue == NULL) {
->        printf("Out of memory!\n");
->        exit(1);
->    }
    queue->head = NULL;
    queue->tail = NULL;
    return queue;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
Queue *createQueue()
{
    Queue *queue = malloc(sizeof(Queue));
    if (queue == NULL) {
        printf("Out of memory!\n");
        exit(1);
    }
    queue->head = NULL;
    queue->tail = NULL;
    return queue;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
static Node *createNode(void *data)
{
    Node *node = malloc(sizeof(Node));
    if (node == NULL) {
        printf("Out of memory!\n");
        exit(1);
    }
    node->data = data;
    node->next = NULL;
    return node;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
-> static Node *createNode(void *data)
{
    Node *node = malloc(sizeof(Node));
    if (node == NULL) {
        printf("Out of memory!\n");
        exit(1);
    }
    node->data = data;
    node->next = NULL;
    return node;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
static Node *createNode(void *data)
-> {
    Node *node = malloc(sizeof(Node));
    if (node == NULL) {
        printf("Out of memory!\n");
        exit(1);
    }
    node->data = data;
    node->next = NULL;
    return node;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
static Node *createNode(void *data)
{
    Node *node = malloc(sizeof(Node));
->    if (node == NULL) {
->        printf("Out of memory!\n");
->        exit(1);
->    }
    node->data = data;
    node->next = NULL;
    return node;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
static Node *createNode(void *data)
{
    Node *node = malloc(sizeof(Node));
    if (node == NULL) {
        printf("Out of memory!\n");
        exit(1);
    }
    node->data = data;
    node->next = NULL;
    return node;
}
/* ... */
```

Queue in C: queue.c

```
/* ... */  
void enqueue(Queue *queue, void *data)  
{  
    Node *node = createNode(data);  
    if (queue->tail == NULL) {  
        queue->head = queue->tail = node;  
    } else {  
        queue->tail->next = node;  
        queue->tail = node;  
    }  
}  
/* ... */
```

Queue in C: queue.c

```
/* ... */
-> void enqueue(Queue *queue, void *data)
{
    Node *node = createNode(data);
    if (queue->tail == NULL) {
        queue->head = queue->tail = node;
    } else {
        queue->tail->next = node;
        queue->tail = node;
    }
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
void enqueue(Queue *queue, void *data)
-> {
    Node *node = createNode(data);
    if (queue->tail == NULL) {
        queue->head = queue->tail = node;
    } else {
        queue->tail->next = node;
        queue->tail = node;
    }
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
void enqueue(Queue *queue, void *data)
{
    Node *node = createNode(data);
->    if (queue->tail == NULL) {
->        queue->head = queue->tail = node;
    } else {
        queue->tail->next = node;
        queue->tail = node;
    }
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
void enqueue(Queue *queue, void *data)
{
    Node *node = createNode(data);
    if (queue->tail == NULL) {
        queue->head = queue->tail = node;
    } else {
        queue->tail->next = node;
        queue->tail = node;
    }
}
/* ... */
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;

        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
-> void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;

        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;
->    if (queue->head == NULL) {
->        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;
        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;
        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;
        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;

        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;

        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
void *dequeue(Queue *queue)
{
    Node *temp;
    void *data;

    if (queue->head == NULL) {
        return NULL;
    } else {
        temp = queue->head;
        queue->head = queue->head->next;

        if (queue->head == NULL) {
            queue->tail = NULL;
        }
        data = temp->data;
        free(temp);
        return data;
    }
}
```

Queue in C: queue.c

```
/* ... */
void deleteQueue(Queue *queue, int freeData)
{
    void *temp;
    do {
        temp = dequeue(queue);
        if (freeData && temp != NULL) {
            free(temp);
        }
    } while (temp != NULL);
    free(queue);
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
void deleteQueue(Queue *queue, int freeData)
{
    void *temp;
->    do {
->        temp = dequeue(queue);
        if (freeData && temp != NULL) {
            free(temp);
        }
->    } while (temp != NULL);
    free(queue);
}
/* ... */
```

Queue in C: queue.c

```
/* ... */
void deleteQueue(Queue *queue, int freeData)
{
    void *temp;
    do {
        temp = dequeue(queue);
        if (freeData && temp != NULL) {
            free(temp);
        }
    } while (temp != NULL);
    free(queue);
}
/* ... */
```

→

Queue in C: queue.c

```
/* ... */
-> void deleteQueue(Queue *queue, int freeData)
{
    void *temp;
    do {
        temp = dequeue(queue);
->         if (freeData && temp != NULL) {
->             free(temp);
->         }
    } while (temp != NULL);
    free(queue);
}
/* ... */
```

Queue in C: queue.c

```
/* ... */  
void deleteQueue(Queue *queue)  
{  
    if (queue->head != NULL) {  
        printf("Queue not empty!");  
        exit(1);  
    }  
    free(queue);  
}  
/* ... */
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    Queue *queue = createQueue();
    /*
        Use the queue here
    */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
-> #include "queue.h"

int main(void) {
    Queue *queue = createQueue();
    /*
        Use the queue here
    */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
→     Queue *queue = createQueue();
        /*
           Use the queue here

        */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    Queue *queue = createQueue();
    /*
```

Use the queue here

```
→     */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c – buggy

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i;
    Queue *queue = createQueue();
    for (i=0;i<100;i++) {
        enqueue(queue, &i);
    }
    /*
    ...

    */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c – buggy

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i;
    Queue *queue = createQueue();
!->    for (i=0;i<100;i++) {
        enqueue(queue, &i);
    }
    /*
        ...

    */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i, *data;
    Queue *queue = createQueue();

    for (i=0;i<100;i++) {
->     data = malloc(sizeof(int));
->     if (data == NULL) {
->         printf("Failed to malloc\n");
->         exit(1);
        }
        *data = i;
        enqueue(queue, data);
    }
    /*
    ...
    */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i, *data;
    Queue *queue = createQueue();

    for (i=0;i<100;i++) {
        data = malloc(sizeof(int));
        if (data == NULL) {
            printf("Failed to malloc\n");
            exit(1);
        }
        *data = i;
        enqueue(queue, data);
    }
    /*
    ...
    */
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i, *data;
    Queue *queue = createQueue();
    /*
     *...
     */
    for (i=0;i<102;i++) {
        data = dequeue(queue);
        if (data != NULL) {
            printf("Dequeued %d\n", *data);
            free(data);
        } else {
            printf("Dequeued NULL\n");
        }
    }
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i, *data;
    Queue *queue = createQueue();
    /*
     *...
     */
    for (i=0;i<102;i++) {
        data = dequeue(queue);
        if (data != NULL) {
            printf("Dequeued %d\n", *data);
            free(data);
        } else {
            printf("Dequeued NULL\n");
        }
    }
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i, *data;
    Queue *queue = createQueue();
    /*
     *...
     */
    for (i=0;i<102;i++) {
        data = dequeue(queue);
        if (data != NULL) {
            printf("Dequeued %d\n", *data);
            free(data);
        } else {
            printf("Dequeued NULL\n");
        }
    }
    deleteQueue(queue, 1);
    return 0;
}
```

Queue in C: main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

int main(void) {
    int i, *data;
    Queue *queue = createQueue();
    /*
     *...
     */
    for (i=0;i<102;i++) {
        data = dequeue(queue);
        if (data != NULL) {
            printf("Dequeued %d\n", *data);
            free(data);
        } else {
            printf("Dequeued NULL\n");
        }
    }
    deleteQueue(queue, 1);
    return 0;
}
```

->

->

->

Compiling

- Compiling a file to an executable consists of three steps:
 1. *Preprocessor*
Processes all 'preprocessor directives'
 2. *Compiler*
Translates the file to machine language
 3. *Linker*
Combines binary files and libraries into executable

Compiling (2)

- For example, compile a program like this:

```
gcc -Wall program.c
```

or

```
cc program.c
```

- All three steps will be performed ...
... resulting an executable file: **a.out**

Compiling (3)

- The compiler usually also accepts multiple files at once:

```
cc file1.c file2.c files3.c
```

- This doesn't really work if you have many files.

Compiling (4)

- Files can also be compiled separately, and then linked:

```
cc -c file1.c  
cc -c file2.c  
cc -c file3.c
```

- The “-c” option makes the compiler skip the link phase.
 - ★ it produces an object file (.o) instead.

Compiling (5)

- The linking phase can then be performed seperately

```
cc file1.o file2.o file3.o -o myprogram
```

- Still doesn't really help with large numbers of files
- Solution: combine object files into a library
 - ★ allows you to re-use the code later

Libraries (1)

- Libraries contain a set of 'utility' functions
 - ★ e.g., I/O, string manipulation, math functions, etc.
- These utility functions can be re-used by
 - ★ including the correct header file
 - ★ linking the library into your executable.

Libraries (2)

- Libraries are usually called “libname.a”
- They can be created with “ar”

```
gcc -c foo.c
```

```
gcc -c bar.c
```

```
ar cr libfoobar.a foo.o bar.o
```

Libraries (3)

- Use “-I...” to specify an include path
- Use “-L...” to specify a library path
- Libraries can be linked with “-lname”
 - ★ just the “name”, skip the “lib” and “.a” parts

```
gcc -I./foobar -L./foobar main.c -lfoobar
```

Libraries (4)

- Use libraries and directories to divide your application into smaller, manageable parts:
 - ★ put “seperate parts” in “seperate directories”
 - ★ create libraries for each of these parts
 - ★ use these libraries in your application
- Problem: doing all this manually is a lot of work

Make (1)

- Solution: make
- Make allows you to “automate” compilation
- Make uses a “Makefile” in every directory
 - ★ contains a description what should be compiled ...
 - ★ ... and how it should be compiled

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

CC = gcc
CFLAGS = -Wall
OBJS = foo.o bar.o
MYLIB = libfoobar.a

all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)

clean:
    rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

→ # 'libfoobar.a' is built from foo.c and bar.c

```
CC = gcc
CFLAGS = -Wall
OBJS = foo.o bar.o
MYLIB = libfoobar.a
```

```
all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)
```

```
clean:
    rm -f *~ *.o *.a
```

→ # ^^^ This space must be a TAB!!.

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

-> CC = gcc
-> CFLAGS = -Wall
-> OBJS = foo.o bar.o
-> MYLIB = libfoobar.a

all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)

clean:
    rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

CC = gcc
CFLAGS = -Wall
OBJS = foo.o bar.o
MYLIB = libfoobar.a

-> all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)

-> clean:
    rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

-> CC = gcc
-> CFLAGS = -Wall
   OBJS = foo.o bar.o
   MYLIB = libfoobar.a

all: $(OBJS)
     ar rc $(MYLIB) $(OBJS)

clean:
     rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

CC = gcc
CFLAGS = -Wall
-> OBJS = foo.o bar.o
-> MYLIB = libfoobar.a

all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)

clean:
    rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

CC = gcc
CFLAGS = -Wall
OBJS = foo.o bar.o
MYLIB = libfoobar.a

-> all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)

-> clean:
    rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

CC = gcc
CFLAGS = -Wall
OBJS = foo.o bar.o
MYLIB = libfoobar.a

-> all: $(OBJS)
->      ar rc $(MYLIB) $(OBJS)

clean:
    rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (1)

```
# 'libfoobar.a' is built from foo.c and bar.c

CC = gcc
CFLAGS = -Wall
OBJS = foo.o bar.o
MYLIB = libfoobar.a

all: $(OBJS)
    ar rc $(MYLIB) $(OBJS)

-> clean:
->     rm -f *~ *.o *.a

# ^^^ This space must be a TAB!!.
```

Makefile (2)

- Make a program like this:

```
make      or      make all
```

- Cleanup like this:

```
make clean
```

- A hierarchy of makefiles can be used for more complex applications

Makefile (3)

```
# 'myprogram' is built from main.c and libfoobar.a

INCLUDE_DIR = ./foobar
LIB_DIR = ./foobar
PROG = myprogram
CC = gcc
CFLAGS = -Wall -I$(INCLUDE_DIR)
OBJS = main.o

all: library $(OBJS)
    gcc -L$(LIB_DIR) $(OBJS) -lfoobar -o $(PROG)

library:
    cd foobar ; make

clean:
    cd foobar ; make clean
    rm -f *~ *.o *.a $(PROG)

# ^^^^ This space must be a TAB!!.
```

Makefile (3)

```
# 'myprogram' is built from main.c and libfooobar.a

INCLUDE_DIR = ./fooobar
LIB_DIR = ./fooobar
PROG = myprogram
CC = gcc
CFLAGS = -Wall -I$(INCLUDE_DIR)
OBJS = main.o

-> all: library $(OBJS)
        gcc -L$(LIB_DIR) $(OBJS) -lfooobar -o $(PROG)

library:
        cd fooobar ; make

clean:
        cd fooobar ; make clean
        rm -f *~ *.o *.a $(PROG)

# ^^^^ This space must be a TAB!!.
```

Makefile (3)

```
# 'myprogram' is built from main.c and libfooobar.a

INCLUDE_DIR = ./fooobar
LIB_DIR = ./fooobar
PROG = myprogram
CC = gcc
CFLAGS = -Wall -I$(INCLUDE_DIR)
OBJS = main.o

all: library $(OBJS)
      gcc -L$(LIB_DIR) $(OBJS) -lfooobar -o $(PROG)

-> library:
->         cd fooobar ; make

clean:
      cd fooobar ; make clean
      rm -f *~ *.o *.a $(PROG)

# ^^^^ This space must be a TAB!!.
```

Makefile (3)

```
# 'myprogram' is built from main.c and libfoobar.a

INCLUDE_DIR = ./foobar
LIB_DIR = ./foobar
PROG = myprogram
CC = gcc
CFLAGS = -Wall -I$(INCLUDE_DIR)
OBJS = main.o

all: library $(OBJS)
->      gcc -L$(LIB_DIR) $(OBJS) -lfoobar -o $(PROG)

library:
      cd foobar ; make

clean:
      cd foobar ; make clean
      rm -f *~ *.o *.a $(PROG)

# ^^^^ This space must be a TAB!!.
```

Debugging

- Debugging can be done in several ways
 - ★ Compiler options
 - ★ Debugging code
 - ★ Assert
 - ★ Debugger

Compiler Options (gcc)

- The compiler can help you with debugging

<code>-Wall</code>	give all warnings
<code>-pedantic</code>	only accept ANSI-C
<code>-Wundef</code>	complain about wrong <code>#if</code> statements
<code>-Wshadow</code>	check if local and global variables have same name
<code>-g</code>	produce debugging code

Debugging code

```
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    int *x = NULL;

#ifdef DEBUG
    if (x == NULL) {
        printf("ERROR: x == NULL!\n");
        exit(1);
    }
#endif

    *x = 100.0;
    return 0;
}
```

```
gcc -DDEBUG exampleDebug.c
./a.out
ERROR: x == NULL!
```

Assert

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

int main(void) {
    int *x = NULL;

    assert(x != NULL);

    *x = 100.0;
    return 0;
}
```

```
gcc -Wall exampleAssert.c
./a.out
exampleAssert.c:8: main: Assertion 'x != ((void *)0)' failed
Aborted
```

(use 'gcc -DNDEBUG exampleAssert.c' to switch off)

Debugger (1)

- When a program crashes it produces a *core dump*
- With a debugger (like *gdb*) you can then find the bug

```
void foo(int *p) {  
    *p = 100;  
}  
int main(void) {  
    int *x = NULL;  
    foo(x);  
    return 0;  
}
```

```
gcc -Wall -g exampleCore.c  
./a.out  
Segmentation fault (core dumped)
```

Debugger (2)

```
/* run the debugger like this */
gdb a.out core

/* Output */
Copyright 1998 Free Software Foundation, Inc.
...
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x0804835a in foo (p=0x0) at exampleCore.c:5
5          *p = 100;
(gdb)
```

Debugger (3)

```
/* On minix, run the debugger like this */  
dcore -c core -e a.out
```

```
-----  
got a corefile for 386 executable a.out
```

	phys_base	vir_base	length
text segm:	0x18bff000	0x00001000	0x00001000

```
...  
ret addr: _foo+0x6 (0x1086)  
ebp= 0x00301004  
ret addr: _main+0x15 (0x10a3)  
ebp= 0x00301014  
ret addr: crtso+0x58 (0x1078)  
...
```

Debugger (3)

```
/* Optinally, use dis386, and find position 0x1086 */  
dis386 ./a.out
```

```
-----  
foo:F12;:
```

```
_foo:
```

```
exampleCore.c:
```

```
    push    ebp                ! 0x1080: 55  
    mov     ebp, esp          ! 0x1081: 89 e5  
    mov     eax, 8(ebp)       ! 0x1083: 8b 45 08  
    mov     (eax), 100        ! 0x1086: c7 00 64 00 00 00  
    leave  ! 0x108c: c9  
    ret     ! 0x108d: c3  
    .  
    .
```

The End

Enjoy!

`http://www.cs.vu.nl/~jason`

`http://www.cs.vu.nl/~cn`