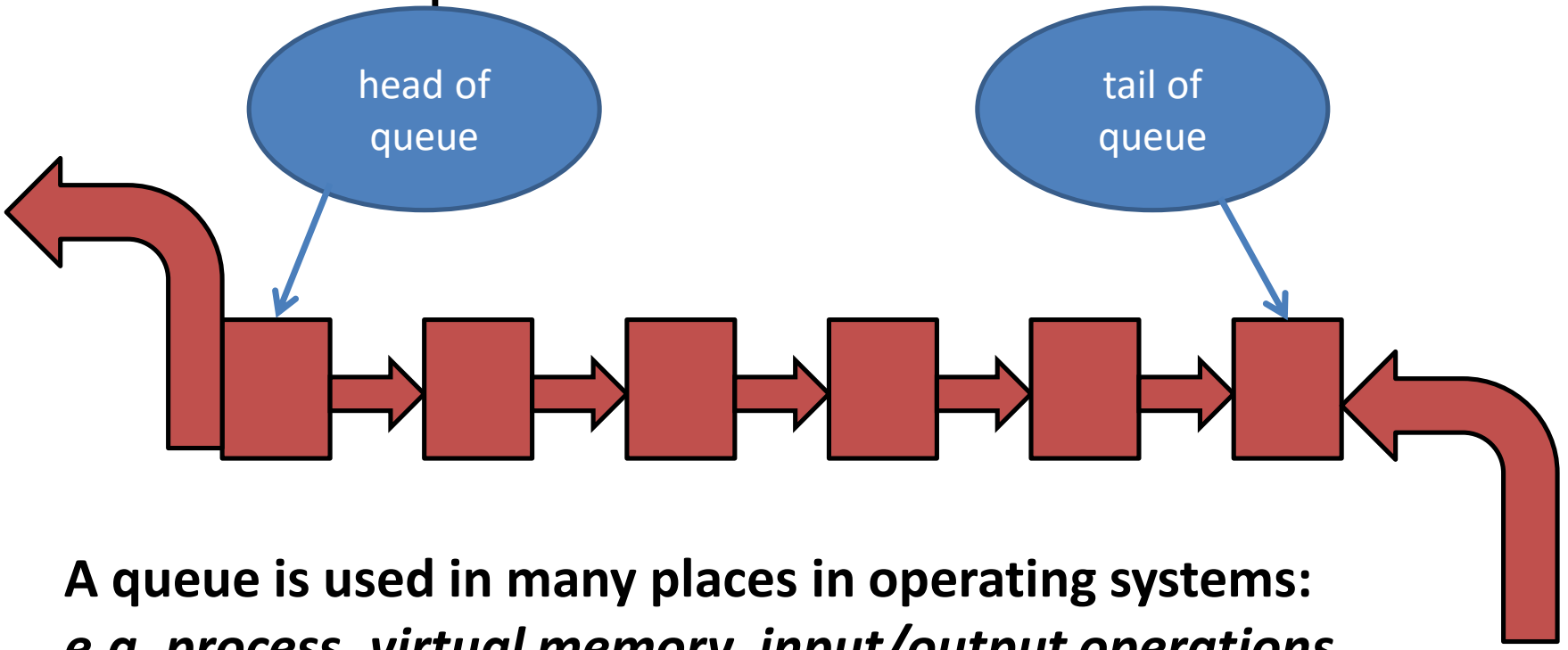
Queues

First in First out subset of a link list

Queue: Data Structures

- What is a queue?



A queue is used in many places in operating systems:
e.g. process, virtual memory, input/output operations
It's a structure that conforms to the principle of First In, First Out (FIFO).
The first item to join the queue is the first item to be served.

The queue

- The queue, like the link list and stack, is comprised of nodes.

- It uses two functions which, by convention, are referred to as:
  - **enqueue** (add node to end of queue)
  - **dequeue** (remove node from head of queue)

- It has the constraint of *not removing* a node from an *empty queue*.

Function Main

denis.manley@apollo: ~/OS2/Week5_stack_queue

```
// function main begins program execution
int main(void)
{
   QueueNode* headPtr = NULL; // initialize headPtr
   QueueNode* tailPtr = NULL; // initialize tailPtr
   char item; // char input by user

   instructions(); // display the menu
   printf("%s", "? ");
   unsigned int choice; // user's menu choice
   scanf("%u", &choice);

   // while user does not enter 3
   while (choice != 3) {

      switch(choice) {
         // enqueue value
         case 1:
            printf("%s", "Enter a character: ");
            scanf("\n%c", &item);
            enqueue(&headPtr, &tailPtr, item);
            printQueue(headPtr);
            break;
         // dequeue value
         case 2:
            // if queue is not empty
            if (headPtr != NULL) {
               item = dequeue(&headPtr, &tailPtr);
               printf("%c has been dequeued.\n", item);
            }

            printQueue(headPtr);
            break;
         default:
```

24,1 16%

Enqueue function

denis.manley@apollo: ~/OS2/Week5_stack_queue

```
void enqueue(QueueNode* *hPtr, QueueNode* *tPtr, char value)
{
   QueueNode* newPtr;

    newPtr= malloc(sizeof(QueueNode));

   if (newPtr != NULL) { // is space available
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      // if empty, insert node at head
      if (*hPtr == NULL) {
        *hPtr = newPtr;
      }
      else {
         (*tPtr)->nextPtr = newPtr;
      }

      *tPtr = newPtr;
   }
   else {

      printf("%c not inserted. No memory available.\n", value);
   }
}
```

78,1 55%

Enqueue 3 nodes: Queue.c

```
dmanleycork@cloudshell:~/Stacks_Queues$ ./Queue
the address of headPtr is: 0x7fff500f9ae8
the address of tailPtr is: 0x7fff500f9af0
the contents of headPtr empty Queue: (nil)
the contens  of tailPtr empty Queue: (nil)
Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 1
Enter a character: R
 ******************  enqueue function ************************
        the contents of headPtr after adding a node  is: 0x5c2d4c6b4ac0
        the contents of tailPtr after adding a node is: 0x5c2d4c6b4ac0
The queue is:
R|(nil) (0x5c2d4c6b4ac0) --> NULL

Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 1
Enter a character: A
 ******************  enqueue function ************************
        the contents of headPtr after adding a node  is: 0x5c2d4c6b4ac0
        the contents of tailPtr after adding a node is: 0x5c2d4c6b4ae0
The queue is:
R|0x5c2d4c6b4ae0 (0x5c2d4c6b4ac0) --> A|(nil) (0x5c2d4c6b4ae0) --> NULL

Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 1
Enter a character: D
 ******************  enqueue function ************************
        the contents of headPtr after adding a node  is: 0x5c2d4c6b4ac0
        the contents of tailPtr after adding a node is: 0x5c2d4c6b4b00
The queue is:
R|0x5c2d4c6b4ae0 (0x5c2d4c6b4ac0) --> A|0x5c2d4c6b4b00 (0x5c2d4c6b4ae0) --> D|(nil) (0x5c2d4c6b4b00) --> NULL
```

Illustrate add node N address 4b20 to this queue

Sample **dequeue** code

denis.manley@apollo: ~/OS2/Week5_stack_queue

```
// remove node from queue head
char dequeue(QueueNode* *hPtr, QueueNode* *tPtr)
{

   char value = (*hPtr)->data;

  /*
        The error prevention code (headPtr != NULL)
        is in function main however
        you can also put it here
   */

   QueueNode* tempPtr = *hPtr;
   *hPtr = (*hPtr)->nextPtr;

   // if queue is empty
   if (*hPtr == NULL) {
      *tPtr = NULL;
   }

   free(tempPtr);
   return value;
                                                    104,1         72%
```

Dequeue 2 nodes: Queue.c

```
The queue is:
R|0x5c2d4c6b4ae0 (0x5c2d4c6b4ac0) --> A|0x5c2d4c6b4b00 (0x5c2d4c6b4ae0) --> D|0x5c2d4c6b4b20 (0x5c2d4c6b4b00) --> N|(nil) (0x5c2d4c6b4b20) --> NULL

Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 2
 ******************  dequeue function ************************
        the contents of headPtr after removing a node is: 0x5c2d4c6b4ae0
        the contents of tailPtr after removing a node is: 0x5c2d4c6b4b20
R has been dequeued.
The queue is:
A|0x5c2d4c6b4b00 (0x5c2d4c6b4ae0) --> D|0x5c2d4c6b4b20 (0x5c2d4c6b4b00) --> N|(nil) (0x5c2d4c6b4b20) --> NULL

Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 2
 ******************  dequeue function ************************
        the contents of headPtr after removing a node is: 0x5c2d4c6b4b00
        the contents of tailPtr after removing a node is: 0x5c2d4c6b4b20
A has been dequeued.
The queue is:
D|0x5c2d4c6b4b20 (0x5c2d4c6b4b00) --> N|(nil) (0x5c2d4c6b4b20) --> NULL
```

4 nodes in Queue. Sample output Deleted (Dequeued) two nodes.

Print Function

```
// print the queue
void printQueue(QueueNode* currentPtr)
{
   // if queue is empty
   if (currentPtr == NULL) {
      puts("Queue is empty.\n");
   }
   else {
      puts("The queue is:");

      // while not end of queue
      while (currentPtr != NULL) {
         printf("%c --> ", currentPtr->data);
         currentPtr = currentPtr->nextPtr;
      }

      puts("NULL\n");
   }
}
```

Why is currentPtr a “single” pointer to a node; What does it store?

Class Question Illustrate the variables in Print Queue

- Given a queue with 3 nodes (refer to slide enqueue to non empty queue) Illustrate:
- show the changes to currentPtr as the list is printed.
-

```
Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 2
 ******************  dequeue function *************************
        the contents of headPtr after removing a node is: 0x564abcd5bae0
        the contents of tailPtr after removing a node is: 0x564abcd5bb20
R has been dequeued.
The queue is:
A|0x564abcd5bb00 (0x564abcd5bae0) --> D|0x564abcd5bb20 (0x564abcd5bb00) --> N|(nil) (0x564abcd5bb20) --> NULL
```

Why does a missing * cause erratic behaviour

```
// insert a node at queue tail
void enqueue(QueueNode* *headPtr, QueueNode* *tailPtr, char value)
{
   QueueNode* newPtr;

    newPtr= malloc(sizeof(QueueNode));

   if (newPtr != NULL) { // is space available
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      // if empty, insert node at head
      if (*headPtr == NULL) {
        headPtr = newPtr;
      }
      else {
         (*tailPtr)->nextPtr = newPtr;
      }

      *tailPtr = newPtr;
   }
   else {

      printf("%c not inserted. No memory available.\n", value);
   }
}
```


Issue with code is here.

Explain why it results below

- Compiling the code gives the following warning and execution of code in previous slide: only showing enqueue function
-

```
dmanleycork@cloudshell:~/Stacks_Queues$ gcc -o QueueClassQuestion QueueClassQuestion.c


QueueClassQuestion.c: In function 'enqueue':
QueueClassQuestion.c:90:17: warning: assignment to 'QueueNode **' {aka 'struct queueNode **'} from
incompatible pointer type 'QueueNode *' {aka 'struct queueNode *'} [-Wincompatible-pointer-types]
   90 |         headPtr = newPtr;
      |                 ^
dmanleycork@cloudshell:~/Stacks_Queues$ ./QueueClassQuestion


Enter your choice:
   1 to add an item to the queue
   2 to remove an item from the queue
   3 to end
? 1
Enter a character: w
Queue is empty.

? 1
Enter a character: G
Queue is empty.
```

Lab/Class Questions

- Hint:
- A warning is not the same as an error. A program will run with a warning but:
- may not behave as expected.
- Cause segmentation fault is often the result in trying to access a memory location that does not exist.

Sample Questions

- Describe, using examples, what is a queue data structure and give two examples of where it may used in operating systems.

Sample question

- Explain, using examples (include node addresses), how to enqueue and dequeue nodes from of a queue **(10 marks)**

- Explain, using examples (include node addresses), how to push and pop functions of a stack **(10 marks)**

Explain if following “enqueue” function will correctly add (enqueue) nodes to a queue **(10 marks)**

denis.manley@apollo: ~/OS2/Week5_stack_queue

```
void enqueue(QueueNode* *headPtr, QueueNode* *tailPtr, char value)
{
   QueueNode* newPtr;

    newPtr= malloc(sizeof(QueueNode));

   if (newPtr != NULL) { // is space available
      newPtr->data = value;
      newPtr->nextPtr = NULL;

      // if empty, insert node at head
      if (*headPtr == NULL) {
         *headPtr = newPtr;
      }
      else {
         (*tailPtr)->nextPtr = newPtr;
      }

      tailPtr = newPtr;
   }
   else {

      printf("%c not inserted. No memory available.\n", value);
   }
}
                                                    90,9          55%
```

Explain what the following code does and ìf it will successfully remove nodes from queue **(10 marks)**

```
char dequeue(QueueNode** headPtr, QueueNode** tailPtr)
{
  if (*headPtr != NULL){
  char value = (*headPtr)->data;
  QueueNode* tempPtr = headPtr;
  headPtr = (*headPtr)->nextPtr;

  if (*headPtr == NULL) {
    tailPtr = NULL;
  }

  free(tempPtr);
  return value;
}
Else
  – Printf("bla bla bla….");
  }
```

Illustrate output

- Illustrate the expected output if you run the above programs: if the Queue has 3 nodes; (5 marks)

- If the code is empty and/or the code is not empty (5 marks)