CS 201
Dynamic Data Structures (3)

Debzani Deb

Overview

- Grading
- Exam 2 review
- Queue
- Binary Tree

Grading

- Exam 2: Highest 47, average 28.
- How Final grading will be done?
  - Labs : 50% (Do not assume linear distribution)
  - Quiz: 2% each (Last one is coming…)
  - Exam 1 & 2 : Each 11% (Look for the scaled grades)
  - Final Exam: 20%
- Special offer: There will be an exam next week
  - Syllabus same as exam 2, Standard same as exam 2.
  - You have the option of replacing your Exam 1 & 2 grade with the result of that single test.
  - Can appear only if you have the probability of getting a F or D.
  - Email me with the intent within next Wednesday.

Exam 2 : Review

Exam 2:Question 1.a

```
4: double average(int *array);
6: int main(void)
7: {
8:          int numberOfMeasurements, i;
9:          int *measurements;
10:         double avg;
12:         printf("Number of measurements?\n");
13:         scanf("%d", &numberOfMeasurements);
15:         measurements = (int *)malloc(numberOfMeasurements);
17:         for(i=0; i<numberOfMeasurements; i++) {
18:                 scanf("%d", measurements + i);
19:         }
21:         avg = average(&measurements);
23:         printf("Average of the measurements: %lf\n", avg);
25:         return 0;
26:}
```

void *malloc(size_t number_of_bytes) size_t argument type is defined in stdlib.h and is an ***unsigned type***

sizeof operator returns the size in bytes of its operand

- If you compile, there will be no syntax error, only one warning
  code.c:21: warning: passing arg 1 of `average' from incompatible pointer type
- The code won't produce desired output.

Exam 2 : Question 1.a

```
28: double average(int *array)
29: {
30:   int i, sum;
31:   double avg;
32:
33:   for(i=0; i<sizeof(array); i++) {
34:       sum += array[i];
35:   }
36:   avg = sum / sizeof(array);
37:
38:   return avg;
39:}
```

```
28: double average(int *array, int size)
29: {
30:   int i, sum;
31:   double avg;
32:
33:   for(i=0; i<size; i++) {
34:       sum += array[i];
35:   }
36:   avg = (double) sum / size;
37:
38:   return avg;
39:}
```

CORRECTED

It is never possible, using sizeof, to find out how long an array a pointer points to; you *must* have a genuine array name instead.

```
int ar[5];
int *ip = ar;
printf("sizeof(ar) is %d\n", sizeof(ar));
printf("sizeof(ip) is %d\n", sizeof(ip));
```

Exam 2 : Question 2.a

- Write a function **Occurrence** which takes two arguments, a string and a character, and returns the number of times the character appears in the string. For example, Occurrence ("Senselessness", 's') should return 6.

```
int Occurrence (char * str, char c) {
  int i, count = 0;
  for (i = 0 ; str[i] != '\0' ; i++)
       if (str[i] == c) count++ ;
  return(count) ;
}
```

- Common mistakes
  - Use of sizeof(str) to determine the string length. WRONG!
    - Use strlen(str) instead
  - Use of NULL instead of '\0' to check the end of the string.

Exam 2 : Question 2.c

Describe the flaw in the following function.

```
/*
* Forms the plural of noun by adding an 's'.
*/
char * add_s(const char *noun) {
  char result[100];
  strcpy(result, noun);
  strcat(result, "s");
  return (result);
}
```

- **Answer:** It returns the address of a locally declared array as the function value. This space is deallocated as soon as the function returns.

Exam 2 : Question 4.a

Given the following type and variable declarations,

```
typedef union {
  char guardian[25];
  char employer[35];
} contact_t;

typedef struct {
  int age;
  contact_t contact;
} person_t;

person_t newcomer;
```

- What memory will be allocated for variable newcomer?
  Answer: one integer and 35 characters

Exam 2 : Question 5.a

Consider this fragment of a C program.

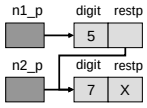
```
int main(void) {
  char *stringp;
  stringp = (char *)calloc(10, sizeof(char));
}
```

- In which portion of the memory, variable stringp and the 10-element array of characters will be allocated?
  Answer: stringp: Stack,
  10-element char array: Heap

Linked List basics (Again)

- A linked list is a sequence of nodes in which each node but the last contains the address of the next node.

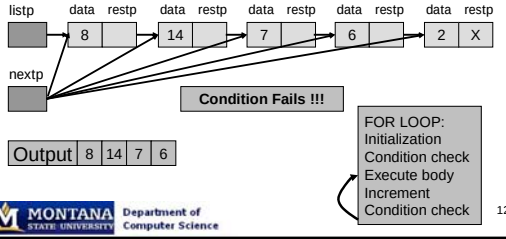
```
typedef struct list_node_s {
  int digit;
  struct list_node_s *restp;
} list_node_t;
list_node_t *n1_p, *n2_p;
n1_p = (list_node_t *) malloc (sizeof(list_node_t));
n2_p = (list_node_t *) malloc (sizeof(list_node_t));
n1_p -> digit = 5;
n2_p -> digit = 7;
n1_p -> restp = n2_p;
n2_p -> restp = NULL;
```



Exam 2 : Question 5.b

- If listp is a pointer to the first node of a linked list containing the data 8 14 7 6 2, what is displayed by the following code fragment?

```
for (nextp = listp; nextp->restp != NULL; nextp = nextp->restp)
  printf("%4d",nextp->data);
```



Exam 2 : Question 5.c

- Write a function display_list that takes a parameter of type node_t * and displays the data from each linked list element on a separate line.

```
void display_list(node_t *listp)
{
    if (listp != NULL) {
            printf("%4d\n", listp->data);
            display_list(listp->restp);
    }
}
```

- Common mistake

```
for (nextp = listp; nextp->restp != NULL; nextp = nextp->restp)
    printf("%4d",nextp->data);
```

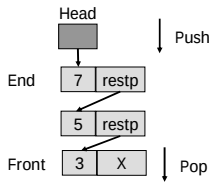

Exam 2 : Question 5.d

- insertLast (Dlist listp, double val), inserts a double number as the last element of DList representing list of double numbers.
  - Recursively/iteratively find the last node of the list.
  - Allocate memory for the new node, copy val to the data field, assign NULL to the pointer field (as last node) and then connect the new node to the list.

Queue

Representing a Queue with a linked list

- Like a queue of people waiting
- Push at the Head (i.e at the end of the list).
- Pop from the Bottom (i.e from the front of the list)
- First In First Out (FIFO)



Push on Queue

```
typedef struct list_node_s {
    int digit;
    struct list_node_s *restp;
} list_node_t;
```

Return pointer to the new head of the queue

Pointer to the current head of the queue

- Push is same as stack (at Head)

```
list_node_t * push (list_node_t * qHead, int v) {
    list_node_t * p = (list_node_t *)
    malloc(sizeof(list_node_t));
    p -> digit = v;
    p -> restp = qHead;
    return p;
}
```

Return the new node as the head of the queue

```
int main(void) {
  list_node_t * qHead = NULL;
  /* Function call for push*/
  qHead = push(qHead, 3);
  qHead = push(qHead, 5);
  return 0;
}
```


Pop from Queue (from the bottom)

```
list_node_t * pop (list_node_t * qHead, int * v) {
    list_node_t * qEnd, * qFront = NULL;
    if (qHead -> restp = NULL) {  // Queue has only one element
      *v = qHead ->digit;
       free (qHead);
       return NULL;
    }
    for (qEnd = qHead; qEnd ->restp != NULL; qEnd = qEnd -> restp)
        qFront = qEnd;
    *v = qEnd -> digit;
     qFront -> restp = NULL;
     free(qEnd);
     return qHead;
}
```

Can we write this more efficiently?

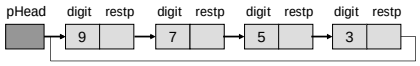
Queue (Summary)

- You could implement it as an array too.
- You could make a hybrid of stack/queue to access at either end.
- Common design for process scheduling, event processing, buffering, input/output etc.
- In our design push is constant time, but pop is O(n) linear time (where n is the number of elements in the queue).
- If we record two pointers (front and end) instead of only one pointer pointing to the head of the list – both push and pop would have constant time.
  - See the implementation in your textbook.

Circular and double linked list

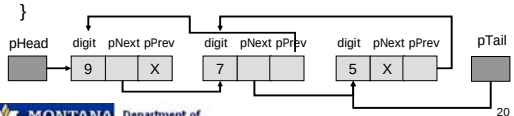
- Circular linked list

```
pHead -> restp -> restp -> restp -> restp = pHead;
```



- Double linked list

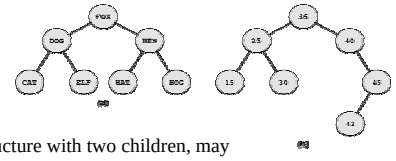
```
struct dblLink {
   int digit;
   struct dblLink * pNext, pPrev;
}
```



Trees

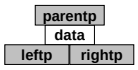
- Non linear data structures
  - Elements can have two or more children.
  - Useful for sorting, graph algorithms, searching etc.
- A simple example – binary tree
  - Each element has 0-2 children.
    - Elements with no children are called a leaf.
  - Each element has 0-1 parents.
    - Elements with no parents are called a root.

Binary Tree



- Structure with two children, may maintain pointer to the parent node too.

```
typedef struct node_s {
   int data;
   struct node_s *leftp, *rightp;
   // struct node_s *parentp;
} node_t;
```



Binary Search Tree

- Useful for sorting
  - Can make an invariant like: all elements in leftp are less than data, all elements in rightp are greater than data.
- Useful for searching
  - With an invariant such as above, you can find an element in the tree in $O(\log_2 n)$ time.
  - Binary search tree is a very common abstraction to maintain.
  - Many algorithms are there.