1 ANSI-C Compiler/Editor Review

1.1 (Talk) Course/Textbook Introduction

1.2 (Talk) The compiler and editor

1.2.1 Installation

1.2.2 Using the editor

1.2.3 Using the compiler

1.3 (In-Class Activity) Use of an editor and compiler

1 **Write a simple program in jEdit**

2 **Compile the program (with different flags check for types of errors)**

```
gcc –c –ansi sdba.c
gcc –c –pedantic sdba.c
gcc –c –Wall sdba.c
```

Compare the error and warning messages you receive.

3 **Link the program from command line.**

```
gcc –o sdba.exe sdba.o
```

4 **Run the program you wrote.**

```
sdba
```

2 Variables and Pointers

2.1 (before class) Variables

Fill in the following table, with fixed (2 decimal places) and floating point calculations (3 figures in mantissa, 1 figure in the exponent)

Equation	Hand Calculation		"Exact"
	Fixed (2 dec)	Floating (3+1)	
133 + 0.22			
133 – 0.99			
13.3 * 1.2			

2.2 (before Class) Pointers. Consider the following code:

```
#include <stdio.h>
int main(void){
  int fred, barney, wilma;
  int *pebbles, *bambam;
  int **dino;
  fred = 1;
  barney = 2;
  wilma = 3;
  pebbles = &fred;
  bambam = &barney;
  dino = &bambam;
  return(0);
}
```

2.2.1 (before class) Draw a VP diagram for the code.

2.2.2 (before class) What is the value of…

a) *pebbles b) bambam c) *dino d) wilma + *pebbles

2.2.3 What happens if we write…

i) *bambam = *pebbles;
ii) bambam = pebbles;
iii) *dino = &wilma;

2.2.4 (before class) Which assignments give errors or warnings?

a) fred = &pebbles; b) pebbles = &fred;
c) pebbles = 5; d) fred = 5;
e) dino = 5; f) *dino = 5;
g) **dino = 5; h) *dino = *pebbles;
i)*dino = pebbles; j) *dino = &pebbles;

(Remember to test on a computer)

3 Pointers and Arrays

3.1 (in-class) Convert the arrays to pointers

```
void strcat(char s[], char t[]){
  int i,j;
  i = 0;
  while (s[i]) i++;
  j = 0;
  while(t[j]){
    s[i] = t[j];
    j++;
    i++;
   }
  s[i]=0;
}
```

3.2 (before class) Solve using an array.

ABC insurance company pays its salespeople a basic salary of $100/week + commission of 10 percent of their gross sales for that week. For example, a salesperson that generates $2000 in policies in a week receives $100 plus 10 percent of $2000, for a total of $300. Write a C program that allows the user to enter the sales of 5 salespeople and outputs a histogram showing how many employees earned salaries in each of the following ranges:

- $100.00-$299.99
- $200.00-$299.99
- $300.00-$399.99
- $400.00-$499.99
- $500.00-$599.99
- $600.00-$699.99
- Over $700.00

The following is a sample user screen that you should try to generate.

```
Input salesperson[1] sales>>
Keyboard Input:     2000
…
Input salesperson[5] sales>>
Keyboard Input:     4500
                Salary              Number
                $100.00-$199.99      0
                $200.00-$299.99      1
…
                $600.00-$699.99      0
                Over $700.00         0
```

4 Records

4.1 (before class) Enter the following code into your computer

```
/* recTut.c :  to help us to learn record access */
#include <stdio.h>
struct customer{
  char lastName[15];
  int customerNumber;
  struct{
    char city[15];
    char zipCode[6];
  } personal;
} customerRecord, *customerPtr;
int main(void){
  struct customer customerRecord=
     {"Ee",7,{"Malacca","75200" } };
  customerPtr = &customerRecord;
/* place printf statements here */
  return(0);
}
```

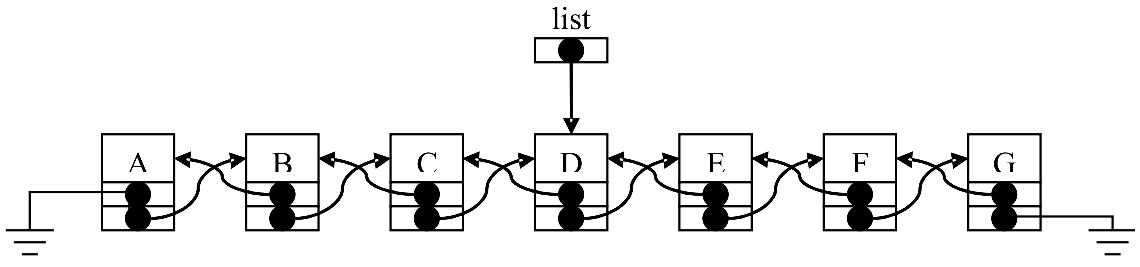
4.2 (before class) Add computer code to print each member 2x. Access by both customerRecord and *customerPtr.

1. Member lastName
2. Member customerNumber
3. Member city of member personal
4. Member zipCode of member personal

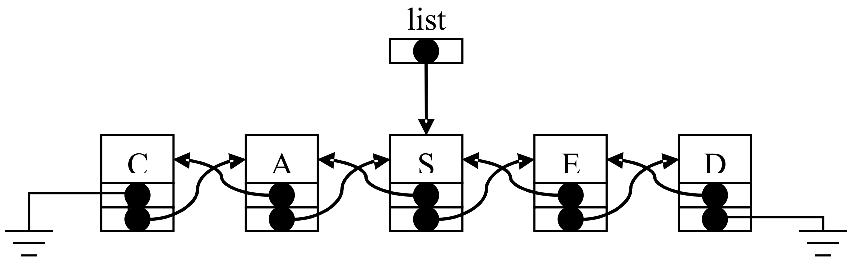
4.3 (in-class) Modify "struct customer" by omitting its internal structure, i.e. struct { ....}personal

5 Creation of a Doubly Linked List

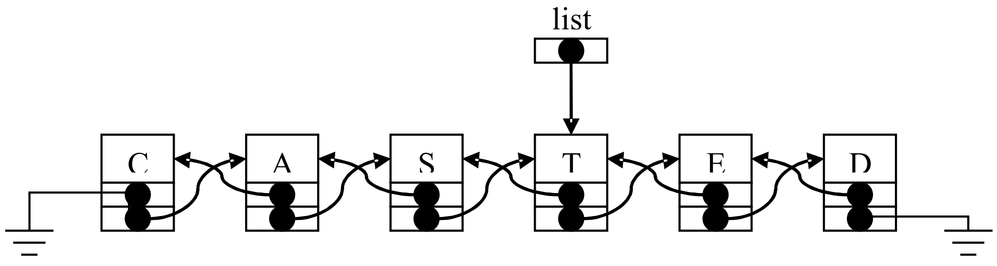
Given a doubly-linked list that looks like this:



In a linked list one needs to have a function to create a new node and connect it to the right of list. For example, if we have:



and we called insertLL(list,'T'), we would get



It would need to take care of the special cases of an empty list (i.e. list is pointing to NULL), and the special case of if list is pointing to the last node.

5.1 (before class) Draw VP diagrams to show the sequence that you would have to go through in order to implement insertLL().

5.2 (in-class) Implement insertLL() in C, and try it out. Don't forget the special cases!

6 Recursion: The Knap Sack Problem

A thief robbing a safe finds it filled with items of varying size and value. Unfortunately, he only has a small knapsack to carry the loot. The problem is to find the combination of items that the robber should choose in order to maximize the total value of all the stolen items.

For example, if the robber has a knapsack of size=20 and the safe contains

item	description	value (US$1000)	size
1	gold bar	5	3
2	silver bar	5	9
3	diamond ring	3	1
4	necklace	2	4
5	Chinese RMB	4	3
6	US$100 bills	7	5
7	Ancient China	8	8
8	gold broach	4	4
9	pendant	1	1

The robber needs to decide what to take and what to leave behind.

In a recursive solution, each time we choose an item, we assume that we can find an optimal way to pack the rest of the knapsack.

6.1 (before class) Write a structure (named item) to hold the name, value and size of an item.

6.2 (before class) Write ANSI-C code to read the items in the safe from a file and store in an array of the structure item.

6.3 (before class) Rewrite the algorithm in ANSI-C.

```
int knapsack(int spaceRemainingInTheKnapsack){
  int totalValueOfItems=0;
  for(each item)
      if(there is room to place this item in the knapsack)
          add it to the knapsack.
          Reduce the available space in the knapsack by size of item.
          value = knapsack(spaceRemainingAfterIncludingThisItem)
              + ValueOfThisItem
          if(value > previousMaximumValue)
              Choose to take this item, increase the maximum value.
  return(maximum value);
```

6.4 (in-class) Run your completed code…

7 Time Complexity and Big-OH

There are two ways to determine the time complexity of algorithms -- empirical analysis or by studying the implemented algorithm. In this lesson we will do both…

7.1 (in-class) Time Complexity by Empirical Analysis

For this case we simply:

1. Choose the size of dataset
2. Run the program for this data set.
3. Record the time taken by the program to run.
4. Increasing the data set.
5. GOTO 2.
6. Fit a polynomial or combination of functions to your time=f(data set size)*

*(ie could use Microsoft Excel to plot and help fit).

If the program is dependent on the nature of the input data (and not just the problem size), the above procedure must be repeated for a variety of data sets. This technique provides a good estimation of average time complexity for the program and requires no knowledge of the detailed inner workings of a program or algorithm.

In this first problem, we will estimate the average time complexity of a program for which we only have an executable version. Please download and run the program timecomp.exe. Use the above algorithm to calculate this unknown program's asymptotic time complexity.

7.2 (in-class) Calculate the time complexity and Big-OH.

```
void ian(int n){
  int i;
  for (i=0; i<n; i++)
    printf("%d",i);
}
```

```
int betty(int n){
  if(n>1)
    return (betty(n-1)+n);
  else return(1);
}
```

```
void siewfong(int n){
  int i, j;
  for (i=0; i<n; i++)
    printf("%d",i);
  for (j=n; j>0; j--)
    printf("%d",j);
}
```

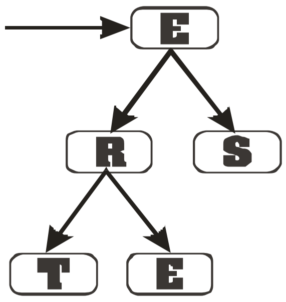
```
void twiddle(int n){
  int i, j;
  for (i=0; i<n; i++)
    for (j=n; j>0; j--)
      printf("%d %d",i,j);
}
```

```
void dino(int n){
  int i, j;
  for (i=0; i<n; i++)
    for (j=0; j<i; j++)
      printf("%d",j);
}
```

8 Discussion of Midterm Paper

9 Trees + Searching (1)

9.1 (before class) Traverse this tree in pre- in- and post-order:



Pre-order:

In-order:

Post-Order:

9.2 (before class) A tree contains 3 number ("5", "6", "2") as nodes along with the mathematical operations (i.e. "*", "+" or "-") Design a tree such that when the calculation is complete, the final answer is "9".

9.3 (before class) Write out an ANSI-C definition for a binary tree node containing a record that contains a person's name and the date of birth

9.4 (in class) Write a function that takes a string as input and checks to see if the string contains a given char.

Use the following calling routine, you supply the function search(). You may not use library functions from <string.h>

```
/* search.c : search for a char in a string */
/* (c)2001 by J D White */
#include <stdio.h>
int search(char *a, char key);
int main(void){
  char key;
  char a[30];
  gets(a);
  key=getchar();
  if(search(a,key))puts("found);
  else puts("Not found");
}
```

10 Searching 2

10.1 (before class) Illustrate step-by-step a linear search for “k”

a	c	e	g	k	l	o	p	q	r	s	t	u	v	w	z

10.2 (before class) Illustrate step-by-step a binary search for “k”

a	c	e	g	k	l	o	p	q	r	s	t	u	v	w	z

10.3 (in-class) Draw a (balanced) binary search tree for “k”

You need only draw the part of the tree one searches on!

10.4 (before class) Place the 7 keys in a hash table of size 9.

Use the modular arithmetic method where “index = key mode size”. Resolve collisions by linear probing. Remember that the ASCII value of 'a' is 97;

Key	**a**	**m**	**g**	**o**	**r**	**i**	**t**		
Value	97								
Modulus	7								
index/step	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1								a	
2									
3									
4									
5									
6									
7									

11 Sorting

11.1 (before class) Use selection

Step\Key	a	l	g	o	r	i	t	h	m
1									
2									
3									
4									
5									
6									
7									
8									
9									

11.2 (before class) Use insertion (underline inserted character)

Key	a	l	g	o	r	i	t	h	m
1									
2									
3									
4									
5									
6									
7									

11.3 (before class) Use merge (add dividers)

Key	a	l	g	o	r	i	t	h	m
1									
2									
3									
4									
5									
6									
7									
8									

12 Quick Sort and Graphs

12.1 (before class) Sort using quick sort

(square brackets around pivot, curved brackets for elements already sorted)

Key	a	l	g	o	r	i	t	h	m
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									

12.2 (before class) Draw the graph for the adjacency matrix

	l	o	v	e	d
l	0	1	3	0	1
o	2	0	1	1	INF
v	1	1	0	0	2
e	3	1	0	0	0
d	1	0	0	1	0

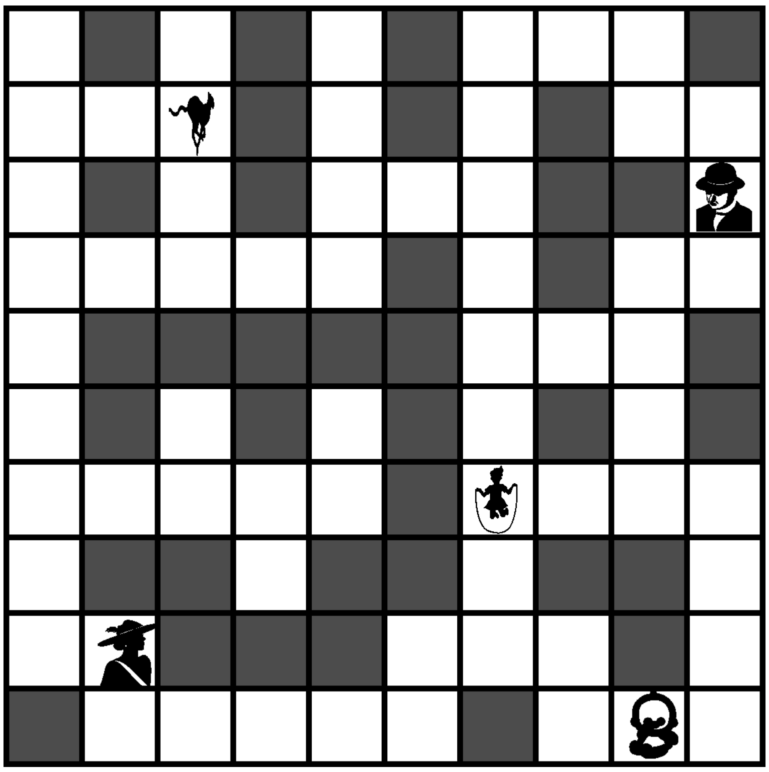
12.3 (before class) What type of graphs is this? (i.e. simple, di-, weighted)

12.4 (before class) Rewrite the adjacency matrix as a list.

12.5 (before class) Traverse the graph in pre- and post-order (alphabetically)

13 Greedy Algorithms

13.1 (before class) Help Xiao-Ming find his wife, Xiu-Man.



13.1.1 Use the backtracking algorithm to move through the maze.

13.1.2 Assume Xiao-ming can hear his wife shouting. Use the greedy algorithm discussed in class to help him find his wife.

13.1.3 Compare the solutions you get using the backtracking algorithm "maze", the greedy algorithm discussed in class, and your own heuristic. The length of the route is the number of squares traveled.

13.1.4 (in class) Determine the shortest path that Xiaoming can travel to reunite his complete family (Xiu-man, Ai-en, You-en and Lightening the cat.)

14 Dijkstra Algorithm

14.1 (before class) Chapter 10 in textbook, question 2

14.2 (before class) Chapter 10 in textbook question 3

14.3 (in class) Use Dijkstra's algorithm to create a minimum spanning tree for the graph LOVED in section 12.2. Start at D and find the shortest distance to each of the other nodes