Midterm #2

1 April, 1994

This is an open books, open notes exam to be turned in by 10:50 AM.

Name: _____

Problem 1. (10 points)

Write a C procedure that takes a pointer to an integer as an argument and adds 15 to the integer. Call the procedure `Add15` and start with the following outline:

```
Add15(int *pI)
  {
   /* You add stuff in here */


  } /* Add15 */
```

Suppose you want to call `Add15` to add 15 to an integer `x`. Fill in the space between the parenthesis in the following statement to show how this is done.

```
Add15(             );
```

Problem 2. (15 points)

Write a C procedure that takes a pointer to a character as an argument and stores the next non-blank character read from standard input into that character. Call the procedure `GetNextRealChar` and start with the following outline:

```
GetNextRealChar(char *pI)
  {









  } /* GetNextRealChar */
```

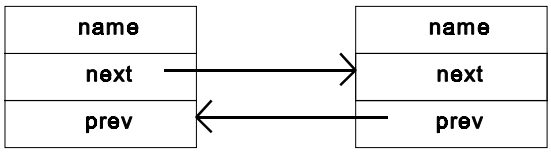
PS: The routine `getchar()` returns the next character read from standard input. `getchar` takes no arguments.

Show how you would call `GetNextRealChar` to read the next non-blank character into the 5'th position of a character array `Buffer`.

```
GetNextRealChar(              ) ;
```

The reaming problems are concerned with doubly linked lists. Use the following C data structure definition in your answers.

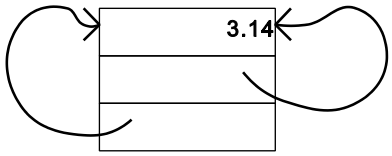
```
typedef struct dl_node_tag {
   float name ;
   struct dl_node *next ;
   struct dl_node *prev ;
} DL_Node_type ;
```



An example of two of these nodes pointing at each other is shown to the left of the C data structure definition.

Problem 3. (15 points)

Write a C procedure that will take a floating point number as an argument and return a single node of type `DL_Node_type` with the `value` field equal to the argument and the `next` and `prev` fields pointing to the newly allocated node. If the procedure is called with the value 3.14, the returned data structure will look like the rather odd node shown on the right.



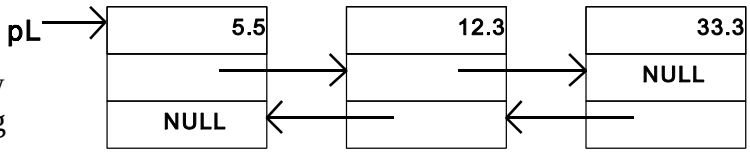

```
DL_Node_type *MakeOddNode(float f)
  {




  } /* MakeOddNode */
```

Problem 4. (20 points)

Starting with the variable `pL` pointing to the leftmost node of the doubly linked list on the right, show the result of executing the following five C statements in order.




```
pL->next->value = 1040.0 ;          /* 1 */
pL->next = pL->next->next ;         /* 2 *
pL->prev = pL->next->prev ;         /* 3 */
pL = pL->next ;                     /* 4 */
pL->next = pL ;                     /* 5 */
```

You may find it useful to use the numbers in the comments to document the changes.

Problem 5. (20 points)

Suppose you have two pointers, `pLeft` and `pRight`, declared as follows:

```
DL_Node_type *pLeft, *pRight;
```

that point to the left and right ends (or head and tail, if you prefer) of a doubly linked list. Write a properly initialized C loop to add up all the floating point numbers stored in the value fields of this doubly linked list and store this `result` in the float variable `sum`.

```
/* Here's some variable declarations to get you started. */
float sum ;
DL_Node_type *pTemp ;
```

Problem 6. (20 points)

Again, suppose you have two pointers, `pLeft` and `pRight`, declared as follows:

```
DL_Node_type *pLeft, *pRight;
```

that point to the left and right ends of a doubly linked list.

This time write a few lines of C to remove the present left and right ends of the list and to set `pLeft` and `pRight` to point to the new ends of the list. Be sure that your code works correctly when the list has a small number, *e. g.*, one or two, of elements.