Since there are few changes between book editions (C has not changed much in 25 years) and to save students money on text books, I allow older editions of the Deitel “C How to Program”. One of the changes in the editions is the numbering of the end of chapter assignments. For example, in the third edition 3.46 is the encode/decode program but in the sixth edition it is 3.49.

For compatibility with my compiler, the following must be used:

#pragma warning(disable: 4996)
#include<string>
#include<stdlib.h>
#include<time.h>
system("pause"); // before return in main

This document gives the assignments independent of the numbering system and page numbers.

Chapter 2:

1) Write a program that inputs two integers and outputs the larger of the two.

2) Write a program that inputs three integers and outputs the smallest, the largest, the sum and the average.

Chapter 3:

Write an encryption and decryption program. Encrypt each digit by adding 7 and taking the remainder after division by 10. After encrypting each digit, swap the first and third then swap the second and fourth. Decryption should reverse the process. Program must input a single integer. Program must do encode and decode in one file.

Example Program Session (yours must look like this):

```
Encode (1) Decode (2): 1
Enter a four digit number: 1234
Encoded Digits: 0189
Continue (1) Exit (0): 1
Encode (1) Decode (2): 2
Enter a four digit number: 0189
Decoded Digits: 1234
Continue (1) Exit (0): 0
```

Chapter 4:

Compute and output compound interest on $1000.00 for 10 years at interest rates of 5%, 6%, 7%, 8%, 9% and 10%

This is an exercise in creating nested loops. You must have an inner loop that calculates compound interest via

For 1 to 10
    amt = rate*amt + amt;

You should have an outer loop that iterates the rate from 0.05 to 1.0.

Interest on $1000.00 over 10 years

rate total
0.05 $1628.89
0.06 $1790.85
0.07 $1967.15
0.08 $2158.92
0.09 $2367.36
0.1 $2593.74

Press any key to continue . . .

Chapter 5:

Write a program that has a function prototype before main and an implementation of the function after main. The function to be implemented is a coin toss simulation using the random number generator. The function should return 1 or true 50% of the time and 0 or false 50% of the time. Use srand and the system time to make the program run differently each time. (`srand(time(NULL));`). Keep track of the number of head and tails for 10, 100, 1000, 10,000, 100,000 and 1,000,000 trials. Output the number of heads and tails and number of each as a percentage of the total. Notice that the more trials the more accurate your simulation becomes.

Example Program Session:

```
Trials: 10
Head count:        6 Percent Heads 60.00
Tail Count:        4 Percent Tails 40.00

Trials: 100
Head count:       57 Percent Heads 57.00
Tail Count:       43 Percent Tails 43.00
```

```
Trials: 1000
Head count:      460 Percent Heads 46.00
Tail Count:      540 Percent Tails 54.00

Trials: 10000
Head count:     4983 Percent Heads 49.83
Tail Count:     5017 Percent Tails 50.17

Trials: 100000
Head count:    50105 Percent Heads 50.10
Tail Count:    49895 Percent Tails 49.90

Trials: 1000000
Head count:   500135 Percent Heads 50.01
Tail Count:   499865 Percent Tails 49.99

Try Again (1) Exit (0)1

Trials: 10
Head count:        2 Percent Heads 20.00
Tail Count:        8 Percent Tails 80.00

Trials: 100
Head count:       43 Percent Heads 43.00
Tail Count:       57 Percent Tails 57.00

Trials: 1000
Head count:      484 Percent Heads 48.40
Tail Count:      516 Percent Tails 51.60

Trials: 10000
Head count:     5005 Percent Heads 50.05
Tail Count:     4995 Percent Tails 49.95

Trials: 100000
Head count:    49881 Percent Heads 49.88
Tail Count:    50119 Percent Tails 50.12

Trials: 1000000
Head count:   499958 Percent Heads 50.00
Tail Count:   500042 Percent Tails 50.00

Try Again (1) Exit (0)0

Press any key to continue . . .
```

Chapter 6:

Research and implement the Sieve of Eratosthenes.

Example Program Session (implement some linefeed '\n' formatting):

```
Enter the limit: 1000
Primes up to 1000

   2    3    5    7   11   13   17   19   23   29   31   37   41   43   47   53
  59   61   67   71   73   79   83   89   97  101  103  107  109  113  127  131
 137  139  149  151  157  163  167  173  179  181  191  193  197  199  211  223
 227  229  233  239  241  251  257  263  269  271  277  281  283  293  307  311
 313  317  331  337  347  349  353  359  367  373  379  383  389  397  401  409
 419  421  431  433  439  443  449  457  461  463  467  479  487  491  499  503
 509  521  523  541  547  557  563  569  571  577  587  593  599  601  607  613
 617  619  631  641  643  647  653  659  661  673  677  683  691  701  709  719
 727  733  739  743  751  757  761  769  773  787  797  809  811  821  823  827
 829  839  853  857  859  863  877  881  883  887  907  911  919  929  937  941
 947  953  967  971  977  983  991  997

Number of primes: 168

Press any key to continue . . .
```

Chapter 7:

Completely describe what the following function does:

```
int mystery( const char *s1, const char *s2 )
{
    for( ; *s1 != '\0' && *s2 != '\0'; s1++, s2++ ) {
                if( *s1 != *s2 ) {
                        return 0;
                }//end if
    }//end for
    return 1;
}
```

Chapter 8:

Write a program that uses the random number generator to create sentences. The program should use four arrays of pointers to char called article, noun, verb and preposition. The sentences are constructed in the following order: article, noun, verb, preposition, article, noun. The program should generate 20 sentences. Capitalize the first letter. The arrays should contain:

```
article "the", "one", "a", "some", "any"
noun "boy", "girl", "dog", "town", "car"
verb "drove", "jumped", "walked", "ran", "skipped"
preposition "to", "from", "over", "under", "on"
```

Example Program Session:

```
A dog skipped to some dog.
One town ran from a girl.
A dog skipped to some dog.
Any car jumped over the car.
One girl jumped on a car.
Any car skipped over the dog.
Some town drove from any town.
A dog skipped to some dog.
A girl skipped from some dog.
A car skipped over the dog.
The boy walked under one boy.
Some town ran from any girl.
The dog walked to a boy.
The boy walked under one boy.
The boy walked under one boy.
One girl ran on a girl.
A dog skipped to some dog.
Any car jumped over the car.
A car skipped over some dog.
A dog skipped to some dog.

Press any key to continue . . .
```

Chapter 9:

Write a program that converts Fahrenheit temperatures to Celsius. Display conversions from 0-212 with 3 digits of precision. Use: `Celsius = 5.0 / 9.0 * (Fahrenheit – 32);`

Example Program Session:

```
F         C

0   -17.778

1   -17.222

2   -16.667

3   -16.111

… repeats
```

```
        203   +95.000
        204   +95.556
        205   +96.111
        206   +96.667
        207   +97.222
        208   +97.778
        209   +98.333
        210   +98.889
        211   +99.444
        212  +100.000
Press any key to continue . . .
```

-

Chapters 10, 12:

This is a two week assignment. Modify the linked list example in the book so that it is a doubly linked list. Prove that your program works properly by implementing a print backwards function. The book's code (6th edition) is at the end of the example output. You can just cut and paste it to get started. NOTE: this is a C program – if you compile it as C++ you cannot use delete as that is a reserved keyword. Also, since C++ is strongly typed, you must cast pointer allocations to the correct type.

You must understand the singly linked list code before you can start the doubly linked list code.

Your code must store in order.

You can redefine Node as:

```
struct listNode {

   char data;

   struct listNode *nextPtr;

   struct listNode *prevPtr;

};
```

If you have problems with improperly assigned pointers, do a more robust printout which shows the node with the address of the data and the pointer values for next and previous.

```
NOTE: I will be testing your code using the following sequence in this exact order –
      Insert b a z k g m
      Delete a z k g b m

This tests the 4 basic cases. Insert/delete on an empty list, insert/delete at the beginning,
insert/delete at the end, insert/delete at the end.
```

Example Program Session:

```
Enter your choice:

   1 to insert an element into the list.

   2 to delete an element from the list.

   3 to end.

? 1

Enter a character: a

The list is:

a --> NULL

The list in reverse is:

a --> NULL
```

```
? 1
Enter a character: z
The list is:
a --> z --> NULL
The list in reverse is:
z --> a --> NULL

? 1
Enter a character: n
The list is:
a --> n --> z --> NULL
The list in reverse is:
z --> n --> a --> NULL

? 1
Enter a character: d
The list is:
a --> d --> n --> z --> NULL
The list in reverse is:
z --> n --> d --> a --> NULL

? 2
Enter character to be deleted: x
x not found.

? 2
Enter character to be deleted: n
n deleted.
The list is:
a --> d --> z --> NULL
The list in reverse is:
z --> d --> a --> NULL
```

```
? 2
Enter character to be deleted: a
a deleted.
The list is:
d --> z --> NULL
The list in reverse is:
z --> d --> NULL

? 2
Enter character to be deleted: z
z deleted.
The list is:
d --> NULL
The list in reverse is:
d --> NULL

? 2
Enter character to be deleted: d
d deleted.
List is empty.

? 1
Enter a character: s
The list is:
s --> NULL
The list in reverse is:
s --> NULL

? 1
Enter a character: t
The list is:
s --> t --> NULL
The list in reverse is:
t --> s --> NULL
```

```
? 3

End of run.

Press any key to continue . . .
```

Here is the code from the book which you can cut and paste. I have modified it to work in C++:

```
#pragma warning(disable: 4996)
#include<string>
#include<stdlib.h>
#include<time.h>
#include<stdio.h>

struct listNode
{
        char data;
        struct listNode *nextPtr;



};

typedef struct listNode ListNode;
typedef ListNode *ListNodePtr;

void insert(ListNode *sPtr, char value);
char erase(ListNodePtr* sPtr, char value);
int isEmpty(ListNode sPtr);
void printList(ListNodePtr currentPtr);
void instructions(void);

int main(void) {
        ListNode startPtr = NULL;
        int choice;
        char item;

        instructions();
        printf("? ");
        scanf("%d", &choice);

        while (choice != 3) {
                switch (choice) {
                case 1:
                        printf("Enter a character: ");
                        scanf("\n%c", &item);
                        insert(&startPtr, item);
                        printList(startPtr);
                        break;

                case 2:
                        if (!isEmpty(startPtr)) {
                                printf("Enter character to be deleted: ");
                                scanf("\n%c", &item);

                                if (erase(&startPtr, item)) {
                                        printf("%c deleted. \n", item);
```

```
                                printList(startPtr);
                        }
                }
                else {
                        printf("%c not found. \n\n", item);

                }

                break;
        default:
                printf("Invalid choice. \n\n");
                instructions();
                break;
        }

        printf("? ");
        scanf("%d", &choice);
        }
    printf("End of run. \n");
    system("pause");
    return 0;

    }
void instructions(void) {
    printf("Enter your choice: \n 1 to insert an element into the list. \n
2 to delete an element from the list. \n 3 to end. \n");


}
void insert(ListNode *sPtr, char value) {
    ListNodePtr newPtr;
    ListNodePtr previousPtr;
    ListNodePtr currentPtr;

    newPtr = malloc(sizeof(ListNode));

    if (newPtr != NULL) {
        newPtr->data = value;
        newPtr->nextPtr = NULL;

        previousPtr = NULL;
        currentPtr = *sPtr;
        while (currentPtr != NULL && value > currentPtr ->data)
        {
                previousPtr = currentPtr;
                currentPtr = currentPtr->nextPtr;

        }
        if (previousPtr == NULL) {
                newPtr->nextPtr = *sPtr;
                *sPtr = newPtr;
        }
        else {
                previousPtr->nextPtr = newPtr;
                newPtr->nextPtr = currentPtr;
        }
```

```
        }
        else {
                printf("%c not inserted. No memory available. \n", value);
        }
}
char erase(ListNodePtr *sPtr, char value) {
        ListNodePtr previousPtr;
        ListNode currentPtr;
        ListNodePtr tempPtr;

        if (value == (*sPtr)->data) {
                tempPtr = *sPtr;
                *sPtr = (*sPtr)->nextPtr;
                free(tempPtr);
                return value;

        }
        else {
                previousPtr = *sPtr;
                currentPtr = (*sPtr)->nextPtr;

                while (currentPtr != NULL && currentPtr->data != value) {
                        previousPtr = currentPtr;
                        currentPtr = currentPtr->nextPtr;

                }
                if (currentPtr != NULL) {
                        tempPtr = currentPtr;
                        previousPtr->nextPtr = currentPtr->nextPtr;
                }

        }

        return '\0';
}
int isEmpty(ListNodePtr sPtr) {
        return sPtr == NULL;
}
void printList(ListNodePtr currentPtr) {
        if (currentPtr == NULL) {
                printf("List is empty. \n\n");
        }
        else {
                printf("The list is:\n");

                while (currentPtr != NULL)
                {
                        printf("%c--> ", currentPtr->data);
                        currentPtr = currentPtr->nextPtr;
                }
                printf("NULL\n\n");
        }
}
```

```
/* Fig. 12.3: fig12_03.c
   Operating and maintaining a list */
```

Chapter 11:

Hardware Inventory – Write a database to keep track of tools, their cost and number. Your program should initialize hardware.dat to 100 empty records, let the user input a record number, tool name, cost and number of that tool. Your program should let you delete and edit records in the database. The next run of the program must start with the data from the last session.

Example Program Session:

(First Run)

```
Enter request
 1 - Input new tool or update an existing tool
 2 - Delete a tool
 3 - List all tools
 4 - Exit
? 3
Record #  Tool name           Quantity  Cost
Enter request
 1 - Input new tool or update an existing tool
 2 - Delete a tool
 3 - List all tools
 4 - Exit
? 1
Enter record number ( 1 to 100, 0 to return to main menu )
? 5
 Enter tool name, quantity, cost
? saw 102 12
Enter record number ( 1 to 100, 0 to return to main menu )
? 7
 Enter tool name, quantity, cost
? hammer 75 8
Enter record number ( 1 to 100, 0 to return to main menu )
? 0
Enter request
 1 - Input new tool or update an existing tool
 2 - Delete a tool
 3 - List all tools
```

```
 4 - Exit

? 3

Record #  Tool name           Quantity  Cost

5         saw                 102       12.00

7         hammer              75        8.00

Enter request

 1 - Input new tool or update an existing tool

 2 - Delete a tool

 3 - List all tools

 4 - Exit

? 4

Press any key to continue . . .
```

(Second Run)

```
Enter request

 1 - Input new tool or update an existing tool

 2 - Delete a tool

 3 - List all tools

 4 - Exit

? 3

Record #  Tool name           Quantity  Cost

5         saw                 102       12.00

7         hammer              75        8.00

Enter request

 1 - Input new tool or update an existing tool

 2 - Delete a tool

 3 - List all tools

 4 - Exit

? 4

Press any key to continue …
```

-

Chapter 13:

Define and use a macro SUMARRAY to sum the values of a numeric array.

-

Chapter 14:

Write a program that calculates the product of a series of integers using a variable-length argument list. Test with several calls, each with a different number of arguments.