Solutions Manual

Chapter 1

1. The code for MS Windows is proprietary and closed source, while the code for many Unix distributions (such as Linux) is free and open source. MS Windows is a single, monolithic, integrated system, while Unix is modular, with users able to switch out pieces of the system. MS Windows is designed to make operating a computer as easy as possible; many details of how the system operates are hidden from users. In a Unix system, all the details are easier to access, for study or for modification.

2. Set a breakpoint to pause execution of the program at a given line number. Print the value of a variable during execution. Run a single line of program code, pausing after it completes.

3. The compiler adds a symbol table to the executable so that variable names from the source code can be understood. The compiler avoids optimizing operations so that lines of code in the executable can be related to the original lines of source code.

4. A text editor can provide a keystroke to move the cursor to a matching brace or parenthesis. It can display the line number, and provide a method to move the cursor to a given line number. It can highlight programming language keywords. It can color code blocks. It can automatically adjust indentation.

5. This is an exploratory problem for the user to test his or her system.

6. This is an exploratory problem for the user to test his or her text editor.

7. The compile-time error is at line #13. It ends in a comma instead of a semicolon. The run-time error is at line #26. The program crashes there when the value of I is zero (the program tries to divide by zero).

8. There are many possible solutions. Here is one:

```
#include <stdio.h>

main()
{
int     n,i;

printf("Enter an integer: ");
scanf("%d",&n);
        /* find i for which i*i <= n and (i+1)*(i+1) >= n */
for (i=0; i<n; i++)
```

```
  if (i*i <= n  &&  (i+1)*(i+1) >= n)
    break;
printf("Closest number having a whole square root: ");
if (n-(i*i) < (i+1)*(i+1)-n)
  printf("%d\n",i*i);
else
  printf("%d\n",(i+1)*(i+1));
}
```

9. There are many possible solutions. Here is one:

```
#include <stdio.h>

main()

{
int     n,sum,last_digit;

printf("Enter an integer: ");
scanf("%d",&n);
sum=0;
while (n > 0)
  {
  last_digit=n%10;
  sum=sum+last_digit;
  n=n/10;
  }
printf("The sum of its digits is %d\n",sum);
}
```

10. There are several possible approaches. Here is one:

    - Figure out how the data will be stored. What will be the variable names, what data types will they use, and how will the program keep track of how much data is stored?
    - Implement the code for displaying all the stored data. This could be tested by hardcoding in values for some of the entries. This way, the storage and display of the data can be tested before any of the other options of the program are implemented.
    - Implement the code for adding a person's data. Test this using the display code implemented previously.
    - Implement the code for deleting a person's data. This should be tested with the option that adds a person's data, on entries at the beginning, middle, and end of the total database.

- Finally, implement and test the code for changing a person's data.

11. The program design is flawed. The program swaps the smallest number currently in the list with the one just entered. This will not maintain a sorted list. The program should be re-designed. One approach is to move the value entered to a position preceding the value that is larger. This can not be done with a simple swap; instead, the rest of the list must be moved forward to make room to insert the new value. If this is done every time a new value is entered, the list will be sorted as intended.

Chapter 2

1. The `double` variable uses 11 bits for the exponent and 52 bits for the fraction; otherwise the pattern is the same as for the `float` variable.

char c=35;	00100011
char d='G';	01000111
int x=-42;	11111111111111111111111111010110
float f=17.25;	01000001100010100000000000000000
int i=1099563008;	01000001100010100000000000000000
double a=17.25;	00001000001100010100000000000000
	00000000000000000000000000000000

2. The following are the two's complement values:

00101101	45
01011010	90
10010001	-111
11100011	-29
0010100010110110	10422
0110111100101011	28459
1100101111001000	-13368
1000000010100011	-32605

3. The problem should read 8 bits for the exponent, not mantissa (this is a typo). Given that change, the answer is that the largest value that can be represented is roughly 2×2^{128}. The smallest fraction that can be represented is roughly 2×2^{-127}.

4. The following are the floating point values:

00111111100000000000000000000000	1.0
01000100100010101110001110001110	1111.111084
11000001000101101011100001010010	-9.42
11000111100010100010011101000000	-70734.5

5. One possible answer is as follows:

```
#include <stdio.h>

main()
{
char n[10];
int x;

printf("Enter a three-digit nonnegative number: ");
scanf("%s",n);
x=(n[0]-'0')*100 + (n[1]-'0')*10 + (n[2]-'0');
```

```
printf("The number is %d\n",x);
}
```

6. The bits could represent anything, depending on what bit model is used to interpret them.

7. The program should use an `int` to store each account identification number. This takes only 4 bytes per number. If an array of `char` were used to store each number, that would take 9 bytes per number.

8. Money should be stored using a whole numer data type, such as an `unsigned char`, `unsigned short int`, or `unsigned int`. This can be done by assuming that each value represents the total number of cents. Using real data types is not necessary, since the fractions of dollars to be stored only take up 2 decimal places.

9. The output of this program is:

```
79 3 122
```

10. The output of this program is:

```
46
```

11. The output of this program is:

```
8
```

12. The output of this program is:

```
1 0
2 2
4 6
8 14
16 14
32 46
64 110
```

13. The output of this program is:

```
100 32 16
90 48 8
80 56 4
70 60 8
60 60 16
50 60 32
40 60 64
30 124 32
20 124 16
10 124 8
```