Chapter 3

\*1. Assume that you want to generate a table of multiples of any given number. Write a program that allows the user to enter the number and then generates the table, formatting it  
into 10 columns and 20 lines. Interaction with the program should look like this (only the  
first three lines are shown):  
Enter a number: 7  
7 14 21 28 35 42 49 56 63 70  
77 84 91 98 105 112 119 126 133 140  
147 154 161 168 175 182 189 196 203 210

Solve

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | #include<iomanip> |
|  | int main() |
|  | { |
|  | int num, i, j, k=1; |
|  |  |
|  | cout <<"Enter a Number For Multiplication: "; |
|  | cin >>num; |
|  | for(i=1; i<=20; i++) |
|  | { |
|  | for(j=1; j<=10; j++) |
|  | { |
|  | cout <<setw(6) <<num\*k++ <<" "; |
|  | } |
|  | cout <<endl; |
|  | } |
|  | return 0; |
|  | } |

\*2. Write a temperature-conversion program that gives the user the option of converting  
Fahrenheit to Celsius or Celsius to Fahrenheit. Then carry out the conversion. Use  
floating-point numbers. Interaction with the program might look like this:  
Type 1 to convert Fahrenheit to Celsius,  
2 to convert Celsius to Fahrenheit: 1  
Enter temperature in Fahrenheit: 70  
In Celsius that’s 21.111111

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | float f,c,temp; |
|  |  |
|  | cout <<"Type 1 to convert temperature from Fahrenheit to Celsius "<<endl |
|  | <<"\nType 2 to convert temperature from Celsius to Fahrenheit: "; |
|  | cin >>temp; |
|  |  |
|  | if(temp==1) |
|  | { |
|  | cout <<"\nEnter Temperature in Fahrenheit: "; |
|  | cin >>f; |
|  | float result = (f-32)\*5/9; |
|  | cout <<"\nIn Celsius that's "<<result; |
|  | } |
|  | else |
|  | { |
|  | cout <<"\nEnter Temperature in Celsius: "; |
|  | cin >>c; |
|  | float result = (c\*9/5)+32; |
|  | cout <<"\nIn Fahrenheit that's "<<result; |
|  | } |
|  | cout <<endl; |
|  | return 0; |
|  | } |

\*3. Operators such as >>, which read input from the keyboard, must be able to convert a  
series of digits into a number. Write a program that does the same thing. It should allow  
the user to type up to six digits, and then display the resulting number as a type long  
integer. The digits should be read individually, as characters, using getche().  
Constructing the number involves multiplying the existing value by 10 and then adding  
the new digit. (Hint: Subtract 48 or ‘0’ to go from ASCII to a numerical digit.)

Here’s some sample interaction:  
Enter a number: 123456  
Number is: 123456

Solve:

| #include<iostream> |
| --- |
|  | #include<conio.h> |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | char ch; |
|  | long total=0; |
|  | cout <<"Enter a number: "; |
|  | for(int i=1; i<=6; i++) |
|  | { |
|  | ch=getche(); |
|  | total=(total\*10)+ch-'0'; |
|  | } |
|  | cout <<"\nNumber is "<<total; |
|  | return 0; |
|  | } |

\*4. Create the equivalent of a four-function calculator. The program should ask the user to  
enter a number, an operator, and another number. (Use floating point.) It should then  
carry out the specified arithmetical operation: adding, subtracting, multiplying, or dividing the two numbers. Use a switch statement to select the operation. Finally, display the  
result.  
When it finishes the calculation, the program should ask whether the user wants to do  
another calculation. The response can be ‘y’ or ‘n’. Some sample interaction with the  
program might look like this:  
Enter first number, operator, second number: 10 / 3  
Answer = 3.333333  
Do another (y/n)? y  
Enter first number, operator, second number: 12 + 100  
Answer = 112  
Do another (y/n)? n

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | float num1,num2; |
|  | char opr, ch; |
|  |  |
|  | do { |
|  | cout <<"\nEnter first number, operator, second number: "; |
|  | cin >>num1 >>opr >>num2; |
|  |  |
|  | switch(opr) |
|  | { |
|  | case '+': |
|  | cout <<"\nAnswer = " <<num1+num2; |
|  | break; |
|  |  |
|  | case '-': |
|  | cout <<"\nAnswer = " <<num1-num2; |
|  | break; |
|  |  |
|  | case '\*': |
|  | cout <<"\nAnswer = " <<num1\*num2; |
|  | break; |
|  |  |
|  | case '/': |
|  | cout <<"\nAnswer = " <<num1/num2; |
|  | break; |
|  |  |
|  | default: |
|  | cout <<"\nInvalid Input! "; |
|  | } |
|  | cout <<"\n\nDo You Want Another, Press(y/n): "; |
|  | cin >>ch; |
|  | }while(ch != 'n'); |
|  | return 0; |
|  | } |

5. Use for loops to construct a program that displays a pyramid of Xs on the screen. The  
pyramid should look like this  
X  
XXX  
XXXXX  
XXXXXXX  
XXXXXXXXX  
except that it should be 20 lines high, instead of the 5 lines shown here. One way to do  
this is to nest two inner loops, one to print spaces and one to print Xs, inside an outer  
loop that steps down the screen from line to line.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | for(int i=1; i<=20; i++) |
|  | { |
|  | for(int j=20; j>=i; j--) |
|  | { |
|  | cout <<" "; |
|  | } |
|  | for(int k=1; k<2\*i; k++) |
|  | { |
|  | cout <<"\*"; |
|  | } |
|  | for(int l=20; l>=i; l--) |
|  | { |
|  | cout <<" "; |
|  | } |
|  | cout <<endl; |
|  | } |
|  | return 0; |
|  | } |

6. Modify the FACTOR program in this chapter so that it repeatedly asks for a number and  
calculates its factorial, until the user enters 0, at which point it terminates. You can  
enclose the relevant statements in FACTOR in a while loop or a do loop to achieve this  
effect.

Solve:

| #include<iostream> |
| --- |
|  | #include<conio.h> |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | long num; |
|  | while(num!=0) |
|  | { |
|  | int fact=1; |
|  | cout <<"\nEnter A Number: "; |
|  | cin >>num; |
|  | for(int i=1; i<=num; i++) |
|  | { |
|  | fact\*=i; |
|  | } |
|  | if(fact!=1) |
|  | cout <<"Factorial of "<<num <<" is "<<fact; |
|  | } |
|  | return 0; |
|  | } |

7. Write a program that calculates how much money you’ll end up with if you invest an  
amount of money at a fixed interest rate, compounded yearly. Have the user furnish the  
initial amount, the number of years, and the yearly interest rate in percent. Some interaction with the program might look like this:  
Enter initial amount: 3000  
Enter number of years: 10  
Enter interest rate (percent per year): 5.5  
At the end of 10 years, you will have 5124.43 dollars.

At the end of the first year you have 3000 + (3000 \* 0.055), which is 3165. At the end of  
the second year you have 3165 + (3165 \* 0.055), which is 3339.08. Do this as many  
times as there are years. A for loop makes the calculation easy

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | int year; |
|  | float amount, interest\_rate, compund\_amount; |
|  |  |
|  | cout <<"\nEnter initial amount: "; |
|  | cin >>amount; |
|  | cout <<"\nEnter number of years: "; |
|  | cin >>year; |
|  | cout <<"\nEnter interest rate (percent per year): "; |
|  | cin >>interest\_rate; |
|  |  |
|  | for(int i=1; i<=year; i++) |
|  | { |
|  | compund\_amount = amount+(amount \* (5.5/100)); |
|  | amount = compund\_amount; |
|  | } |
|  | cout <<"\nAt the end of "<<year <<" years, you will have "<<compund\_amount <<" dollars." <<endl; |
|  | return 0; |
|  | } |

8. Write a program that repeatedly asks the user to enter two money amounts expressed in  
old-style British currency: pounds, shillings, and pence. (See Exercises 10 and 12 in  
Chapter 2, “C++ Programming Basics.”) The program should then add the two amounts  
and display the answer, again in pounds, shillings, and pence. Use a do loop that asks the  
user whether the program should be terminated. Typical interaction might be  
Enter first amount: £5.10.6  
Enter second amount: £3.2.6  
Total is £8.13.0  
Do you wish to continue (y/n)?  
To add the two amounts, you’ll need to carry 1 shilling when the pence value is greater  
than 11, and carry 1 pound when there are more than 19 shillings.

Solve;

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | int res\_pound, res\_shilling, res\_pence, pound1, pound2, shilling1, shilling2, pence1, pence2; |
|  | char ch; |
|  |  |
|  | do{ |
|  | cout <<"\nEnter first amount: "; |
|  | cin >>pound1 >>shilling1>> pence1; |
|  |  |
|  | cout <<"\nEnter second amount: "; |
|  | cin >>pound2 >>shilling2 >> pence2; |
|  |  |
|  | res\_pound = pound1 + pound2; |
|  | res\_shilling = shilling1 + shilling2; |
|  | res\_pence = pence1 + pence2; |
|  |  |
|  | if(res\_pence > 11) |
|  | { |
|  | res\_shilling++; |
|  | res\_pence = 0; |
|  | } |
|  | if(res\_shilling > 19) |
|  | { |
|  | res\_pound++; |
|  | } |
|  | cout <<"\nTotal is " <<res\_pound <<"." <<res\_shilling <<"." <<res\_pence; |
|  |  |
|  | cout <<"\nDo you wish to continue (y/n): "; |
|  | cin >>ch; |
|  | } while(ch != 'n'); |
|  |  |
|  | return 0; |
|  | } |

9. Suppose you give a dinner party for six guests, but your table seats only four. In how  
many ways can four of the six guests arrange themselves at the table? Any of the six  
guests can sit in the first chair. Any of the remaining five can sit in the second chair. Any  
of the remaining four can sit in the third chair, and any of the remaining three can sit in  
the fourth chair. (The last two will have to stand.) So the number of possible arrangements of six guests in four chairs is 6\*5\*4\*3, which is 360. Write a program that calculates the number of possible arrangements for any number of guests and any number of chairs. (Assume there will never be fewer guests than chairs.) Don’t let this get too complicated. A simple for loop should do it.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | int guests, chairs, temp=1; |
|  | cout <<"\nEnter the number of guests: "; |
|  | cin >>guests; |
|  | cout <<"\nEnter the number of chairs: "; |
|  | cin >>chairs; |
|  |  |
|  | if(guests>chairs) |
|  | { |
|  | for(int i=1; i<=chairs; i++) |
|  | { |
|  | temp \*= guests; |
|  | --guests; |
|  | } |
|  | } |
|  | cout <<"\nNumber of possible arrangements are: " <<temp; |
|  |  |
|  | return 0; |
|  | } |

10. Write another version of the program from Exercise 7 so that, instead of finding the final  
amount of your investment, you tell the program the final amount and it figures out how  
many years it will take, at a fixed rate of interest compounded yearly, to reach this  
amount. What sort of loop is appropriate for this problem? (Don’t worry about fractional  
years; use an integer value for the year.)

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | float amount, prinicap\_amount, rat\_interest, temp; |
|  | int year; |
|  | cout <<"Enter the final amount you want to get in particular year: "; |
|  | cin >>amount; |
|  | cout <<"Enter the principal amount: "; |
|  | cin >>prinicap\_amount; |
|  | cout <<"Enter the rate of interest: "; |
|  | cin >>rat\_interest; |
|  |  |
|  | temp = prinicap\_amount; |
|  |  |
|  | while(temp<amount) |
|  | { |
|  | temp \*= (1+(rat\_interest/100)); |
|  | year++; |
|  | } |
|  | cout <<"It will take " <<year <<" years at " <<rat\_interest <<" of interest to reach the final amount " <<amount <<endl; |
|  | return 0; |
|  | } |

11. Create a three-function calculator for old-style English currency, where money amounts  
are specified in pounds, shillings, and pence. (See Exercises 10 and 12 in Chapter 2.)  
The calculator should allow the user to add or subtract two money amounts, or to multiply a money amount by a floating-point number. (It doesn’t make sense to multiply two  
money amounts; there is no such thing as square money. We’ll ignore division. Use the  
general style of the ordinary four-function calculator in Exercise 4 in this chapter.)

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | float pound1, pound2, shilling1, shilling2, pence1, pence2; |
|  | char option; |
|  | cout <<"\nEnter first amount in old style english currency: \x9C"; |
|  | cin >>pound1 >>shilling1 >>pence1; |
|  |  |
|  | cout <<"\nEnter second amount in old style english currency: \x9C"; |
|  | cin >>pound2 >>shilling2 >>pence2; |
|  |  |
|  | cout <<"\nPress '+' for addition, press '-' for subtraction & press '\*' for multiplication: "; |
|  | cin >>option; |
|  |  |
|  | switch(option) { |
|  | case '+': |
|  | cout <<"Addition of two old style english currency is: \x9C" <<pound1+pound2 <<"." <<shilling1+shilling2 <<"." <<pence1+pence2 <<endl; |
|  | break; |
|  |  |
|  | case '-': |
|  | cout <<"subtraction of two old style english currency is: \x9C" <<pound1-pound2 <<"." <<shilling1-shilling2 <<"." <<pence1-pence2 <<endl; |
|  | break; |
|  |  |
|  | case '\*': |
|  | cout <<"Multiplication of two old style englsih currency is: \x9C" <<pound1\*pound2 <<"." <<shilling1\*shilling2 <<"." <<pence1\*pence2 <<endl; |
|  | break; |
|  |  |
|  | default: |
|  | cout <<"Invalid Option!"; |
|  |  |
|  | } |
|  | return 0; |
|  | } |

12. Create a four-function calculator for fractions. (See Exercise 9 in Chapter 2, and  
Exercise 4 in this chapter.) Here are the formulas for the four arithmetic operations  
applied to fractions:  
Addition: a/b + c/d = (a\*d + b\*c) / (b\*d)  
Subtraction: a/b - c/d = (a\*d - b\*c) / (b\*d)  
Multiplication: a/b \* c/d = (a\*c) / (b\*d)  
Division: a/b / c/d = (a\*d) / (b\*c)  
The user should type the first fraction, an operator, and a second fraction. The program  
should then display the result and ask whether the user wants to continue.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  | int main() |
|  | { |
|  | float a, b, c, d; |
|  | char option; |
|  | cout <<"\nEnter the first fraction: "; |
|  | cin >>a >>b; |
|  |  |
|  | cout <<"\nEnter an operator(+, -, \*, /): "; |
|  | cin >>option; |
|  |  |
|  | cout <<"\nEnter the second fraction: "; |
|  | cin >>c >>d; |
|  |  |
|  | switch(option) { |
|  | case '+': |
|  | cout <<"Addition: " <<a<<"/"<<b<<" + "<<c<<"/"<<d <<" = " <<((a\*d+b\*c)/(b\*d)) <<endl; |
|  | break; |
|  |  |
|  | case '-': |
|  | cout <<"Subtraction: "<<a<<"/"<<b<<" - "<<c<<"/"<<d <<" = " <<((a\*d-b\*c)/(b\*d)) <<endl; |
|  | break; |
|  |  |
|  | case '\*': |
|  | cout <<"Multiplication: "<<a<<"/"<<b<<" \* "<<c<<"/"<<d <<" = " <<((a\*c)/(b\*d)) <<endl; |
|  | break; |
|  |  |
|  | case '/': |
|  | cout <<"Division: "<<a<<"/"<<b<<" / "<<c<<"/"<<d <<" = " <<((a\*d)/(b\*c)) <<endl; |
|  | break; |
|  |  |
|  | default: |
|  | cout <<"Invalid Option!"; |
|  | } |
|  | return 0; |
|  | } |

Chapter 5

\*1. Refer to the CIRCAREA program in Chapter 2, “C++ Programming Basics.” Write a function called circarea() that finds the area of a circle in a similar way. It should take an  
argument of type float and return an argument of the same type. Write a main() function that gets a radius value from the user, calls circarea(), and displays the result.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | float circarea(float); |
|  |  |
|  | int main() { |
|  | float rad; |
|  | cout <<"\nEnter radius of a circle: "; |
|  | cin >>rad; |
|  | cout <<"Area of circle is: " <<circarea(rad) <<endl; |
|  | return 0; |
|  | } |
|  |  |
|  | //circarea() |
|  | float circarea(float rads) { |
|  | return 3.14159\*rads\*rads; |
|  | } |

\*2. Raising a number n to a power p is the same as multiplying n by itself p times. Write a  
function called power() that takes a double value for n and an int value for p, and  
returns the result as a double value. Use a default argument of 2 for p, so that if this  
argument is omitted, the number n will be squared. Write a main() function that gets values from the user to test this function.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | double power(double, int=2); |
|  |  |
|  | int main() { |
|  | double num, answer; |
|  | int pow; |
|  | char ch; |
|  | cout <<"\nEnter a number: "; |
|  | cin >>num; |
|  | cout <<"Want to enter the power Y/N: "; |
|  | cin >>ch; |
|  | if(ch == 'y' || ch == 'Y') { |
|  | cout <<"Enter the power: "; |
|  | cin >>pow; |
|  | answer = power(num, pow); |
|  | } else { |
|  | answer = power(num); |
|  | } |
|  | cout <<"The value of power is: " <<answer <<endl; |
|  | return 0; |
|  | } |
|  |  |
|  | //power() |
|  | double power(double n, int power) { |
|  | double ans = 1.0; |
|  | for(int i=1; i<=power; i++) { |
|  | ans \*= n; |
|  | } |
|  | return ans; |
|  | } |

\*3. Write a function called zeroSmaller() that is passed two int arguments by reference  
and then sets the smaller of the two numbers to 0. Write a main() program to exercise  
this function.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | void zeroSmaller(int&, int&); |
|  |  |
|  | int main() { |
|  | int num1, num2; |
|  | cout <<"\nEnter two integer number: "; |
|  | cin >> num1 >>num2; |
|  | zeroSmaller(num1, num2); |
|  | cout <<"Number 1 is: " <<num1 <<endl; |
|  | cout <<"Number 2 is: " <<num2 <<endl; |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | void zeroSmaller(int& a, int& b) { |
|  | if(a>b) |
|  | b=0; |
|  | else |
|  | a=0; |
|  | } |

\*4. Write a function that takes two Distance values as arguments and returns the larger one.  
Include a main() program that accepts two Distance values from the user, compares  
them, and displays the larger. (See the RETSTRC program for hints.)

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | struct Distance { |
|  | int feet; |
|  | float inches; |
|  | }; |
|  |  |
|  | Distance lrgDist(Distance, Distance); |
|  | void lrgdisp(Distance); |
|  |  |
|  | int main() { |
|  | Distance d1, d2, d3; |
|  | cout <<"\nEnter first distance in feet: "; |
|  | cin >>d1.feet; |
|  | cout <<"Enter first distance in inches: "; |
|  | cin >>d1.inches; |
|  | cout <<"\nEnter second distance in feet: "; |
|  | cin >>d2.feet; |
|  | cout <<"Enter second distance in inches: "; |
|  | cin >>d2.inches; |
|  |  |
|  | d3 = lrgDist(d1, d2); |
|  |  |
|  | cout <<"\nd1: "; lrgdisp(d1); |
|  | cout <<"d2: "; lrgdisp(d2); |
|  |  |
|  | cout <<"\nThe largest distance is: "; |
|  | lrgdisp(d3); |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | Distance lrgDist(Distance dd1, Distance dd2) { |
|  | if(dd1.feet > dd2.feet) { |
|  | return dd1; |
|  |  |
|  | } else if(dd2.feet > dd1.feet) { |
|  | return dd2; |
|  |  |
|  | } else if(dd1.inches > dd2.inches) { |
|  | return dd1; |
|  |  |
|  | } else { |
|  | return dd2; |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | void lrgdisp(Distance disp) { |
|  | cout <<disp.feet <<"' -" <<disp.inches <<"\"" <<endl; |
|  | } |

5. Write a function called hms\_to\_secs() that takes three int values—for hours, minutes,  
and seconds—as arguments, and returns the equivalent time in seconds (type long).  
Create a program that exercises this function by repeatedly obtaining a time value in  
hours, minutes, and seconds from the user (format 12:59:59), calling the function, and  
displaying the value of seconds it returns.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | long hms\_to\_secs(int, int, int); |
|  |  |
|  | int main() { |
|  | int hours, minutes, seconds; |
|  | char ch = 'a'; |
|  | do { |
|  | cout <<"\nEnter hours: "; |
|  | cin >>hours; |
|  | cout <<"Enter minutes: "; |
|  | cin >>minutes; |
|  | cout <<"Enter seconds: "; |
|  | cin >>seconds; |
|  | cout <<"\nPress s to submit your values: "; |
|  | cin >>ch; |
|  | } while(ch !='s'); |
|  |  |
|  | cout <<"\nTotal time in seconds is: " <<hms\_to\_secs(hours, minutes, seconds) <<endl; |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | long hms\_to\_secs(int hh, int mm, int ss) { |
|  | long total\_seconds = hh\*3600 + mm\*60 + ss; |
|  | return total\_seconds; |
|  | } |

6. Start with the program from Exercise 11 in Chapter 4, “Structures,” which adds two  
struct time values. Keep the same functionality, but modify the program so that it uses  
two functions. The first, time\_to\_secs(), takes as its only argument a structure of type

time, and returns the equivalent in seconds (type long). The second function,  
secs\_to\_time(), takes as its only argument a time in seconds (type long), and returns a  
structure of type time.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | struct time { |
|  | int hours; |
|  | int minutes; |
|  | int seconds; |
|  | }; |
|  |  |
|  | long time\_to\_secs(time); |
|  | time secs\_to\_time(long); |
|  | void disp\_time(time); |
|  |  |
|  | int main() { |
|  | time t1, str\_time; |
|  | long time; |
|  | cout <<"\nEnter hours: "; |
|  | cin >>t1.hours; |
|  | cout <<"Enter minutes: "; |
|  | cin >>t1.minutes; |
|  | cout <<"Enter seconds: "; |
|  | cin >>t1.seconds; |
|  |  |
|  | cout <<"Total time structure in seconds is: " <<time\_to\_secs(t1); |
|  |  |
|  | cout <<"\n\nEnter total time in seconds: "; |
|  | cin >>time; |
|  | str\_time = secs\_to\_time(time); |
|  | cout <<"Total seconds in time structure is: "; disp\_time(str\_time); |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | long time\_to\_secs(time tt1) { |
|  | return tt1.hours\*3600 + tt1.minutes\*60 + tt1.seconds; |
|  | } |
|  |  |
|  |  |
|  | time secs\_to\_time(long tt2) { |
|  | int hh, mm, ss; |
|  | time tme; |
|  | mm = tt2/60; |
|  | ss = tt2%60; |
|  | hh = mm/60; |
|  | mm = mm%60; |
|  |  |
|  | tme.hours = hh; |
|  | tme.minutes = mm; |
|  | tme.seconds = ss; |
|  |  |
|  | return tme; |
|  | } |
|  |  |
|  |  |
|  | void disp\_time(time tms){ |
|  | cout <<tms.hours <<":" <<tms.minutes <<":" <<tms.seconds <<endl; |
|  | } |

7. Start with the power() function of Exercise 2, which works only with type double.  
Create a series of overloaded functions with the same name that, in addition to double,  
also work with types char, int, long, and float. Write a main() program that exercises  
these overloaded functions with all argument types.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | double power(double, int=2); |
|  | char power(char, int=2); |
|  | int power(int, int=2); |
|  | long power(long, int=2); |
|  | float power(float, int=2); |
|  |  |
|  | int main() { |
|  | double num1; |
|  | char num2; |
|  | int num3; |
|  | long num4; |
|  | float num5; |
|  |  |
|  | cout <<"\nEnter a double vale for power calculation: "; |
|  | cin >>num1; |
|  | cout <<"Power of " <<num1 <<" is: " <<power(num1); |
|  |  |
|  | cout <<"\n\nnEnter a char value for power calculation: "; |
|  | cin >>num2; |
|  | cout <<"Power of " <<num2 <<" is: " <<power(num2); |
|  |  |
|  | cout <<"\n\nEnter a int value for power calculation: "; |
|  | cin >>num3; |
|  | cout <<"Power of " <<num3 <<" is: " <<power(num3); |
|  |  |
|  | cout <<"\n\nEnter a long value for power calculation: "; |
|  | cin >>num4; |
|  | cout <<"Power of " <<num4 <<" is: " <<power(num4); |
|  |  |
|  | cout <<"\n\nEnter a float value for power calculation: "; |
|  | cin >>num5; |
|  | cout <<"Power of " <<num5 <<" is: " <<power(num5); |
|  |  |
|  | cout <<endl; |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | double power(double dd, int p) { |
|  | double power = 1; |
|  | for(int i=1; i<=p; i++) |
|  | power \*= dd; |
|  | return power; |
|  | } |
|  |  |
|  |  |
|  | char power(char ch, int p) { |
|  | char power = 1; |
|  | for(int i=1; i<=p; i++) |
|  | power \*= ch; |
|  | return power; |
|  | } |
|  |  |
|  |  |
|  | int power(int no1, int p) { |
|  | int power = 1; |
|  | for(int i=1; i<=p; i++) |
|  | power \*= no1; |
|  | return power; |
|  | } |
|  |  |
|  |  |
|  | long power(long numb, int p) { |
|  | long power = 1; |
|  | for (int i=1; i<=p; i++) |
|  | power \*= numb; |
|  | return power; |
|  | } |
|  |  |
|  |  |
|  | float power(float number, int p) { |
|  | float power = 1.0; |
|  | for (int i=1; i<=p; i++) |
|  | power \*= number; |
|  | return power; |
|  | } |

8. Write a function called swap() that interchanges two int values passed to it by the calling program. (Note that this function swaps the values of the variables in the calling program, not those in the function.) You’ll need to decide how to pass the arguments. Create  
a main() program to exercise the function.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | void swap(int&, int&); |
|  |  |
|  | int main() { |
|  | int num1, num2; |
|  | cout <<"\nEnter two number for swaping: "; |
|  | cin >>num1 >>num2; |
|  | cout <<"\nnum1 is: " <<num1 <<endl; |
|  | cout <<"num2 is: " <<num2 <<endl; |
|  | swap(num1, num2); |
|  | cout <<"\n\nAfter swap num1 is " <<num1 <<endl; |
|  | cout <<"After swap num2 is " <<num2 <<endl; |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | void swap(int& a, int& b){ |
|  | int temp = a; |
|  | a=b; |
|  | b=temp; |
|  | } |

9. Repeat Exercise 8, but instead of two int variables, have the swap() function interchange two struct time values (see Exercise 6).

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | struct time { |
|  | int hours; |
|  | int minutes; |
|  | int seconds; |
|  | }; |
|  |  |
|  | void swapTime(time&, time&); |
|  | void swapTimeDisp(time); |
|  |  |
|  | int main() { |
|  | time t1, t2; |
|  | cout <<"\nEnter 1st time hours: "; cin >>t1.hours; |
|  | cout <<"Enter 1st time minutes: "; cin >>t1.minutes; |
|  | cout <<"Enter 1st time seconds: "; cin >>t1.seconds; |
|  |  |
|  | cout <<"\n\nEnter 2nd time hours: "; cin >>t2.hours; |
|  | cout <<"Enter 2nd time minutes: "; cin >>t2.minutes; |
|  | cout <<"Enter 2nd time seconds: "; cin >>t2.seconds; |
|  |  |
|  | cout <<"\n\nBefore swap 1st time is: "; swapTimeDisp(t1); |
|  | cout <<"Before swap 2st time is: "; swapTimeDisp(t2); |
|  |  |
|  | swapTime(t1, t2); |
|  |  |
|  | cout <<"\n\nAfter swap 1st time is: "; swapTimeDisp(t1); |
|  | cout <<"After swap 2nd time is: "; swapTimeDisp(t2); |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | void swapTime(time& tm1, time& tm2) { |
|  | time temp; |
|  | temp.hours = tm1.hours; |
|  | temp.minutes = tm1.minutes; |
|  | temp.seconds = tm1.seconds; |
|  |  |
|  | tm1.hours = tm2.hours; |
|  | tm1.minutes = tm2.minutes; |
|  | tm1.seconds = tm2.seconds; |
|  |  |
|  | tm2.hours = temp.hours; |
|  | tm2.minutes = temp.minutes; |
|  | tm2.seconds = temp.seconds; |
|  | } |
|  |  |
|  |  |
|  | void swapTimeDisp(time tme) { |
|  | cout <<tme.hours <<":" <<tme.minutes <<":" <<tme.seconds <<endl; |
|  | } |

10. Write a function that, when you call it, displays a message telling how many times it has  
been called: “I have been called 3 times”, for instance. Write a main() program that calls  
this function at least 10 times. Try implementing this function in two different ways.  
First, use a global variable to store the count. Second, use a local static variable. Which  
is more appropriate? Why can’t you use a local variable?

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | int count=0; |
|  |  |
|  | void globalVar(); |
|  | int localstatic(); |
|  |  |
|  | int main() { |
|  | int store; |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | globalVar(); |
|  | localstatic(); |
|  | localstatic(); |
|  | localstatic(); |
|  | localstatic(); |
|  | localstatic(); |
|  | localstatic(); |
|  | store = localstatic(); |
|  |  |
|  | cout <<"\nI have been called " <<count <<" times using global variable" <<endl; |
|  | cout <<"\nI have been called " <<store <<" times using local static variable" <<endl; |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | void globalVar() { |
|  | count++; |
|  | } |
|  |  |
|  |  |
|  | int localstatic() { |
|  | static int remember = 0; |
|  | remember++; |
|  | return remember; |
|  | } |

11. Write a program, based on the sterling structure of Exercise 10 in Chapter 4, that  
obtains from the user two money amounts in old-style British format (£9:19:11), adds  
them, and displays the result, again in old-style format. Use three functions. The first  
should obtain a pounds-shillings-pence value from the user and return the value as a  
structure of type sterling. The second should take two arguments of type sterling and  
return a value of the same type, which is the sum of the arguments. The third should take  
a sterling structure as its argument and display its value.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | struct sterling { |
|  | int pound; |
|  | int shilling; |
|  | int pence; |
|  | }; |
|  |  |
|  | sterling BritishCurrency(int, int, int); |
|  | sterling sterlingfunc(sterling, sterling); |
|  | void sterlingDisp(sterling); |
|  |  |
|  | int main() { |
|  | int pound, shilling, pence; |
|  | sterling firstValue, secondValue; |
|  | sterling s1 = {2, 10, 5}; |
|  | sterling s2 = {9,18, 10}; |
|  | cout <<"\nEnter pound: "; cin >>pound; |
|  | cout <<"Enter shilling: "; cin >>shilling; |
|  | cout <<"Enter pence: "; cin >>pence; |
|  | firstValue = BritishCurrency(pound, shilling, pence); |
|  | secondValue = sterlingfunc(s1, s2); |
|  | cout <<"\nFirst value is: "; sterlingDisp(firstValue); |
|  | cout <<"\nSecond value is: "; sterlingDisp(secondValue); |
|  | cout <<endl; |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | sterling BritishCurrency(int pnd, int shl, int pen) { |
|  | sterling currency; |
|  | currency.pound = pnd; |
|  | currency.shilling = shl; |
|  | currency.pence = pen; |
|  | return currency; |
|  | } |
|  |  |
|  |  |
|  | sterling sterlingfunc(sterling ss1, sterling ss2) { |
|  | sterling result; |
|  | result.pound = ss1.pound + ss2.pound; |
|  | result.shilling = ss1.shilling + ss2.shilling; |
|  | result.pence = ss1.pence + ss2.pence; |
|  |  |
|  | if(result.pence > 11) { |
|  | result.shilling++; |
|  | result.pence = 0; |
|  | } |
|  |  |
|  | if(result.shilling > 19) { |
|  | reuslt.pound++; |
|  | } |
|  |  |
|  | return result; |
|  | } |
|  |  |
|  |  |
|  | void sterlingDisp(sterling valueDisp) { |
|  | cout <<"\x9c" <<valueDisp.pound <<":" <<valueDisp.shilling <<":" <<valueDisp.pence; |
|  | } |

12. Revise the four-function fraction calculator from Exercise 12, Chapter 4, so that it uses  
functions for each of the four arithmetic operations. They can be called fadd(), fsub(),  
fmul(), and fdiv(). Each of these functions should take two arguments of type struct  
fraction, and return an argument of the same type

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | struct strctValue { |
|  | int numerator; |
|  | int denominator; |
|  | char dummyslash; |
|  | }; |
|  |  |
|  | strctValue fadd(strctValue, strctValue); |
|  | strctValue fsub(strctValue, strctValue); |
|  | strctValue fmul(strctValue, strctValue); |
|  | strctValue fdiv(strctValue, strctValue); |
|  | void strctDisp(strctValue); |
|  |  |
|  | int main() { |
|  | strctValue value1, value2, addshow, subshow, mulshow, divshow; |
|  | char option; |
|  | cout <<"\nEnter value for first fraction in (a/b) format: "; |
|  | cin >>value1.numerator >>value1.dummyslash >>value1.denominator; |
|  |  |
|  | cout <<"Enter value for second fraction in (c/d) format: "; |
|  | cin >>value2.numerator >>value2.dummyslash >> value2.denominator; |
|  |  |
|  | cout <<"\nChoose an operator for performing respective operation (+, -, \*, /): "; |
|  | cin >>option; |
|  |  |
|  | switch(option) { |
|  | case '+': |
|  | addshow = fadd(value1, value2); |
|  | cout <<"\nValue of addition is: "; strctDisp(addshow); |
|  | break; |
|  |  |
|  | case '-': |
|  | subshow = fsub(value1, value2); |
|  | cout <<"\nValue of subtraction is: "; strctDisp(subshow); |
|  | break; |
|  |  |
|  | case '\*': |
|  | mulshow = fmul(value1, value2); |
|  | cout <<"\nValue of multiplication is: "; strctDisp(mulshow); |
|  | break; |
|  |  |
|  | case '/': |
|  | divshow = fdiv(value1, value2); |
|  | cout <<"\nValue of division is: "; strctDisp(divshow); |
|  | break; |
|  |  |
|  | default: |
|  | cout <<"Invalid operator"; |
|  | } |
|  |  |
|  | cout <<endl; |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  | strctValue fadd(strctValue val1, strctValue val2) { |
|  | strctValue add; |
|  | add.numerator = val1.numerator\*val2.denominator + val1.denominator\*val2.numerator; |
|  | add.denominator = val2.denominator\*val2.denominator; |
|  | return add; |
|  | } |
|  |  |
|  |  |
|  | strctValue fsub(strctValue val1, strctValue val2) { |
|  | strctValue sub; |
|  | sub.numerator = val1.numerator\*val2.denominator - val1.denominator\*val2.numerator; |
|  | sub.denominator = val1.denominator\*val2.denominator; |
|  | return sub; |
|  | } |
|  |  |
|  |  |
|  | strctValue fmul(strctValue val1, strctValue val2) { |
|  | strctValue mul; |
|  | mul.numerator = val1.numerator\*val2.numerator; |
|  | mul.denominator = val1.denominator\*val2.denominator; |
|  | return mul; |
|  | } |
|  |  |
|  |  |
|  | strctValue fdiv(strctValue val1, strctValue val2) { |
|  | strctValue div; |
|  | div.numerator = val1.numerator\*val2.denominator; |
|  | div.denominator = val1.denominator\*val2.numerator; |
|  | return div; |
|  | } |
|  |  |
|  |  |
|  | void strctDisp(strctValue disp) { |
|  | cout <<disp.numerator <<"/" <<disp.denominator; |
|  | } |

chapter 6

Answers to the starred exercises can be found in Appendix G.  
\*1. Create a class that imitates part of the functionality of the basic data type int. Call the  
class Int (note different capitalization). The only data in this class is an int variable.  
Include member functions to initialize an Int to 0, to initialize it to an int value, to display it (it looks just like an int), and to add two Int values.  
Write a program that exercises this class by creating one uninitialized and two initialized  
Int values, adding the two initialized values and placing the response in the uninitialized  
value, and then displaying this result.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class Int{ |
|  | private: |
|  | int value; |
|  | public: |
|  | Int():value(0){ |
|  |  |
|  | } |
|  | Int(int val):value(val){ |
|  |  |
|  | } |
|  | void dispValue(); |
|  | void addValue(Int, Int); |
|  | }; |
|  |  |
|  | //dispValue defination |
|  | void Int::dispValue(){ |
|  | cout <<"value is: "<<value; |
|  | } |
|  |  |
|  | //addValue defination |
|  | void Int::addValue(Int num1, Int num2){ |
|  | value = num1.value+num2.value; |
|  | } |
|  |  |
|  | int main(){ |
|  | Int number1; |
|  | Int number2(4); |
|  | Int number3(10); |
|  | number1.addValue(number2, number3); |
|  | cout <<"After addition the "; |
|  | number1.dispValue(); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

\*2. Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 cent  
toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps  
track of the number of cars that have gone by, and of the total amount of money collected.

Model this tollbooth with a class called tollBooth. The two data items are a type  
unsigned int to hold the total number of cars, and a type double to hold the total amount  
of money collected. A constructor initializes both of these to 0. A member function called  
payingCar() increments the car total and adds 0.50 to the cash total. Another function,  
called nopayCar(), increments the car total but adds nothing to the cash total. Finally, a  
member function called display() displays the two totals. Make appropriate member  
functions const.  
Include a program to test this class. This program should allow the user to push one key  
to count a paying car, and another to count a nonpaying car. Pushing the Esc key should  
cause the program to print out the total cars and total cash and then exit.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class tollboth{ |
|  | private: |
|  | unsigned int totalcarNum; |
|  | double totalMoney; |
|  | public: |
|  | tollboth():totalcarNum(0), totalMoney(0){ |
|  | } |
|  | void payingCar(){ |
|  | totalcarNum++; |
|  | totalMoney+=0.50; |
|  | } |
|  | void nopayCar(){ |
|  | totalcarNum++; |
|  | } |
|  | void display() const{ |
|  | cout <<"\nTotal Number of cars passed on the toll both: " <<totalcarNum; |
|  | cout <<"\nTotal Money collected at toll both: " <<totalMoney; |
|  | } |
|  | }; |
|  |  |
|  | int main(){ |
|  | tollboth button; |
|  | char count; |
|  |  |
|  | cout <<"Press 'c' to count paying car and 'm' to count non paying car."; |
|  | cout <<"\nPress ESC to exit from program."; |
|  | cout <<endl <<endl; |
|  | do{ |
|  | cout <<"Press Key: "; |
|  | cin >> count; |
|  | if(count=='c' || count=='C'){ |
|  | button.payingCar(); |
|  | } else if(count=='m' || count=='M'){ |
|  | button.nopayCar(); |
|  | } else { |
|  | if(count == 27) { |
|  | break; |
|  | } else { |
|  | cout <<"Invalid key pressed!\n"; |
|  | } |
|  | } |
|  | }while(count != 27); |
|  |  |
|  | button.display(); |
|  |  |
|  | cout <<endl; |
|  | return 0; |
|  | } |

\*3. Create a class called time that has separate int member data for hours, minutes, and  
seconds. One constructor should initialize this data to 0, and another should initialize it  
to fixed values. Another member function should display it, in 11:59:59 format. The final  
member function should add two objects of type time passed as arguments.  
A main() program should create two initialized time objects (should they be const?) and  
one that isn’t initialized. Then it should add the two initialized values together, leaving the  
result in the third time variable. Finally it should display the value of this third variable.  
Make appropriate member functions const.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class times { |
|  | public: |
|  | int hours; |
|  | int minutes; |
|  | int seconds; |
|  | public: |
|  | times():hours(0), minutes(0), seconds(0) //no-argument constructor |
|  | { } |
|  | times(int hrs, int min, int sec):hours(hrs), minutes(min), seconds(sec) |
|  | { } |
|  | void dispTime() const { |
|  | cout <<hours <<":" <<minutes <<":" <<seconds; |
|  | } |
|  | void addTime(times, times); |
|  | }; |
|  |  |
|  | void times::addTime(times t1, times t2) { |
|  | seconds = t1.seconds + t2.seconds; |
|  | if(seconds > 59) { |
|  | seconds -= 60; |
|  | minutes++; |
|  | } |
|  |  |
|  | minutes += t1.minutes + t2. minutes; |
|  | if(minutes > 59) { |
|  | minutes -= 60; |
|  | hours++; |
|  | } |
|  | hours += t1.hours + t2.hours; |
|  | } |
|  |  |
|  | int main() { |
|  | const times t1(10, 30, 43); |
|  | const times t2(2, 26, 53); |
|  | times t3; |
|  | t3.addTime(t1, t2); |
|  | cout <<"Total time in t3 is: "; |
|  | t3.dispTime(); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

4. Create an employee class, basing it on Exercise 4 of Chapter 4. The member data should  
comprise an int for storing the employee number and a float for storing the employee’s  
compensation. Member functions should allow the user to enter this data and display it.  
Write a main() that allows the user to enter data for three employees and display it.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class employee { |
|  | private: |
|  | int employeeNumber; |
|  | float employeeCompensation; |
|  | public: |
|  | employee(): employeeNumber(0), employeeCompensation(0) |
|  | { } |
|  | employee(int empNum, float empCom): employeeNumber(empNum), employeeCompensation(empCom) |
|  | { } |
|  | void setData(); |
|  |  |
|  | void dispData() const; |
|  | }; |
|  |  |
|  | void employee::setData() { |
|  | cout <<"\nEnter employee Number: "; |
|  | cin >>employeeNumber; |
|  | cout <<"Enter employee Compensation: "; |
|  | cin >>employeeCompensation; |
|  | cout <<endl; |
|  | } |
|  |  |
|  | void employee::dispData() const { |
|  | cout <<"Employee Number: " <<employeeNumber <<endl; |
|  | cout <<"Employee Compensation: $" <<employeeCompensation <<endl; |
|  | } |
|  |  |
|  | int main() { |
|  | employee emp1, emp2; |
|  | employee emp3(12, 34.6); |
|  | emp1.setData(); |
|  | emp2.setData(); |
|  | cout <<"\nDetails of first employee: "; |
|  | emp1.dispData(); |
|  | cout <<"\nDetails of second employee: "; |
|  | emp2.dispData(); |
|  | cout <<"\nDetails of third employee: "; |
|  | emp3.dispData(); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

5. Start with the date structure in Exercise 5 in Chapter 4 and transform it into a date  
class. Its member data should consist of three ints: month, day, and year. It should also  
have two member functions: getdate(), which allows the user to enter a date in  
12/31/02 format, and showdate(), which displays the date.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class dates { |
|  | private: //member datas |
|  | int month; |
|  | int day; |
|  | int year; |
|  | public: //member functions |
|  | void getDate(); |
|  |  |
|  | void showDate(); |
|  | }; |
|  |  |
|  | void dates::getDate() { |
|  | cout <<"Enter date in mm/dd/yy format: "; |
|  | cin >>month >>day >>year; |
|  | } |
|  |  |
|  | void dates::showDate() { |
|  | cout <<"Date: " <<month <<"/" <<day <<"/" <<year; |
|  | } |
|  |  |
|  | int main() { |
|  | dates dt; |
|  | dt.getDate(); |
|  | dt.showDate(); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

6. Extend the employee class of Exercise 4 to include a date class (see Exercise 5) and an  
etype enum (see Exercise 6 in Chapter 4). An object of the date class should be used to  
hold the date of first employment; that is, the date when the employee was hired. The  
etype variable should hold the employee’s type: laborer, secretary, manager, and so on.  
These two items will be private member data in the employee definition, just like the  
employee number and salary. You’ll need to extend the getemploy() and putemploy()  
functions to obtain this new information from the user and display it. These functions will  
probably need switch statements to handle the etype variable. Write a main() program that  
allows the user to enter data for three employee variables and then displays this data.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | enum etype {laborer, secretary, manager, accountant, executive, researcher  } |
|  |  |
|  | class dates { |
|  | private: |
|  | int day; |
|  | int month; |
|  | int year; |
|  | public: |
|  | void putDate(); |
|  | void showDate(); |
|  | }; |
|  |  |
|  | void dates::putDate() { |
|  | char ch; |
|  | cout <<"Enter employee date of joining in dd/mm/yyyy format: "; |
|  | cin >>day >>ch >>month >>ch >>year; |
|  | } |
|  |  |
|  | void dates::showDate() { |
|  | cout <<"\nEmployee date of joining: "<<day <<"/" <<month <<"/" <<year; |
|  | } |
|  |  |
|  | class employee { |
|  | private: |
|  | int employeeNumber; |
|  | float employeeCompensation; |
|  | dates dt; |
|  | etype emp; |
|  | public: |
|  | void putemploy(); |
|  | void getemploy(); |
|  | }; |
|  |  |
|  | void employee::putemploy() { |
|  | char selectempType; |
|  | cout <<"Enter employee number: "; |
|  | cin >>employeeNumber; |
|  | cout <<"Enter employee Compensation: "; |
|  | cin >>employeeCompensation; |
|  | dt.putDate(); |
|  | cout <<"Enter employee type (first letter only: laborer, secretary, manager, accountant, executive, researcher): "; |
|  | cin >>selectempType; |
|  |  |
|  | switch(selectempType) { |
|  | case 'l': |
|  | emp = laborer; |
|  | break; |
|  |  |
|  | case 's': |
|  | emp = secretary; |
|  | break; |
|  |  |
|  | case 'm': |
|  | emp = manager; |
|  | break; |
|  |  |
|  | case 'a': |
|  | emp = accountant; |
|  | break; |
|  |  |
|  | case 'e': |
|  | emp = executive; |
|  | break; |
|  |  |
|  | case 'r': |
|  | emp = researcher; |
|  | break; |
|  |  |
|  | default: |
|  | cout <<"Invalid Input!"; |
|  | } |
|  | } |
|  |  |
|  | void employee::getemploy() { |
|  | cout <<"\nEmployee Number: " <<employeeNumber; |
|  | cout <<"\nEmployee Compensation $" <<employeeCompensation; |
|  | dt.showDate(); |
|  | cout <<"\nEmployee type is: "; |
|  | switch(emp) { |
|  | case laborer: |
|  | cout <<"Laborer"; |
|  | break; |
|  |  |
|  | case secretary: |
|  | cout <<"Secretary"; |
|  | break; |
|  |  |
|  | case manager: |
|  | cout <<"Manager"; |
|  | break; |
|  |  |
|  | case accountant: |
|  | cout <<"Accountant"; |
|  | break; |
|  |  |
|  | case executive: |
|  | cout <<"Executive"; |
|  | break; |
|  |  |
|  | case researcher: |
|  | cout <<"Researcher"; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | int main() { |
|  | employee emp1, emp2, emp3; |
|  | cout <<"\nEnter first employee details: \n"; |
|  | emp1.putemploy(); |
|  | cout <<"\nEnter second employee details: \n"; |
|  | emp2.putemploy(); |
|  | cout <<"\nEnter third employee details: \n"; |
|  | emp3.putemploy(); |
|  | cout <<"\nFirst employee deatils: "; |
|  | emp1.getemploy(); |
|  | cout <<endl; |
|  | cout <<"\nSecond employee deatils: "; |
|  | emp2.getemploy(); |
|  | cout <<endl; |
|  | cout <<"\nThird employee deatils: "; |
|  | emp3.getemploy(); |
|  | cout <<endl <<endl; |
|  | return 0; |
|  | } |

7. In ocean navigation, locations are measured in degrees and minutes of latitude and longitude. Thus if you’re lying off the mouth of Papeete Harbor in Tahiti, your location is 149  
degrees 34.8 minutes west longitude, and 17 degrees 31.5 minutes south latitude. This is

written as 149°34.8’ W, 17°31.5’ S. There are 60 minutes in a degree. (An older system  
also divided a minute into 60 seconds, but the modern approach is to use decimal minutes  
instead.) Longitude is measured from 0 to 180 degrees, east or west from Greenwich,  
England, to the international dateline in the Pacific. Latitude is measured from 0 to 90  
degrees, north or south from the equator to the poles.  
Create a class angle that includes three member variables: an int for degrees, a float  
for minutes, and a char for the direction letter (N, S, E, or W). This class can hold either  
a latitude variable or a longitude variable. Write one member function to obtain an angle  
value (in degrees and minutes) and a direction from the user, and a second to display the  
angle value in 179°59.9’ E format. Also write a three-argument constructor. Write a  
main() program that displays an angle initialized with the constructor, and then, within a  
loop, allows the user to input any angle value, and then displays the value. You can use  
the hex character constant ‘\xF8’, which usually prints a degree (°) symbol.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class angle { |
|  | private: |
|  | int degrees; |
|  | float minutes; |
|  | char direction; |
|  | public: |
|  | angle():degrees(0), minutes(0), direction(0) |
|  | { } |
|  | angle(int deg, float min, char dir):degrees(deg), minutes(min), direction(dir) //three arguments constructor |
|  | { } |
|  | void putAngle(int, float, char); |
|  | void dispAngle() const; |
|  | }; |
|  |  |
|  | void angle::putAngle(int dgr, float mnt, char dir) { |
|  | degrees = dgr; |
|  | minutes = mnt; |
|  | direction = dir; |
|  | } |
|  |  |
|  | void angle::dispAngle() const{ |
|  | cout <<"Angle: " <<degrees <<"\xF8 " <<minutes <<"\' " <<direction; |
|  |  |
|  | switch(direction) { |
|  | case 'N': |
|  | cout <<" Latitude"; |
|  | break; |
|  | case 'n': |
|  | cout <<" Latitude"; |
|  | break; |
|  | case 'S': |
|  | cout <<" Latitude"; |
|  | break; |
|  | case 's': |
|  | cout <<" Latitude"; |
|  | break; |
|  | case 'E': |
|  | cout <<" Longitude"; |
|  | break; |
|  | case 'e': |
|  | cout <<" Longitude"; |
|  | break; |
|  | case 'W': |
|  | cout <<" Longitude"; |
|  | break; |
|  | case 'w': |
|  | cout <<" Longitude"; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | int main() { |
|  | int degr; |
|  | float mint; |
|  | char drct, choice; |
|  | cout <<endl; |
|  | angle angle1(146, 34.6, 'S'); |
|  | angle1.dispAngle(); |
|  | cout <<endl; |
|  | angle angle2; |
|  |  |
|  | do { |
|  | cout <<"\nEnter the degree: "; |
|  | cin >>degr; |
|  | cout <<"Enter the minutes: "; |
|  | cin >>mint; |
|  | cout <<"Enter the direction(N, S, E, W): "; |
|  | cin >>drct; |
|  |  |
|  | angle2.putAngle(degr, mint, drct); |
|  | angle2.dispAngle(); |
|  | cout <<"\n\nDo you want to enter another angle for navigation (Y, N): "; |
|  | cin >>choice; |
|  | } while(choice == 'Y' || choice == 'y' ); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

8. Create a class that includes a data member that holds a “serial number” for each object  
created from the class. That is, the first object created will be numbered 1, the second 2,  
and so on.  
To do this, you’ll need another data member that records a count of how many objects  
have been created so far. (This member should apply to the class as a whole; not to  
individual objects. What keyword specifies this?) Then, as each object is created, its  
constructor can examine this count member variable to determine the appropriate serial  
number for the new object.  
Add a member function that permits an object to report its own serial number. Then  
write a main() program that creates three objects and queries each one about its serial  
number. They should respond I am object number 2, and so on.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class serailnumberCount { |
|  | private: |
|  | static int count; |
|  | unsigned int serialNumber; |
|  | public: |
|  | serailnumberCount() { |
|  | count++; |
|  | serialNumber = count; |
|  | } |
|  | void dispserailNumber() const; |
|  | }; |
|  |  |
|  | int serailnumberCount::count = 0; |
|  |  |
|  | //function defination |
|  | void serailnumberCount::dispserailNumber() const { |
|  | cout <<"I am object number " <<serialNumber; |
|  | } |
|  |  |
|  | int main() { |
|  | serailnumberCount obj1, obj2, obj3; |
|  | obj1.dispserailNumber(); |
|  | cout <<endl; |
|  | obj2.dispserailNumber(); |
|  | cout <<endl; |
|  | obj3.dispserailNumber(); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

9. Transform the fraction structure from Exercise 8 in Chapter 4 into a fraction class.  
Member data is the fraction’s numerator and denominator. Member functions should  
accept input from the user in the form 3/5, and output the fraction’s value in the same  
format. Another member function should add two fraction values. Write a main() program  
that allows the user to repeatedly input two fractions and then displays their sum. After  
each operation, ask whether the user wants to continue.

Solve:

| #include<iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class addFraction { |
|  | private: |
|  | int numerator; |
|  | int denominator; |
|  | public: |
|  | addFraction():numerator(0), denominator(0) //no argument constructor |
|  | { } |
|  | void getFraction(); |
|  | void add(addFraction, addFraction); |
|  | void dispFraction(); |
|  | }; |
|  |  |
|  | void addFraction::getFraction() { |
|  | char dummyChar; |
|  | cin >>numerator >>dummyChar >>denominator; |
|  | } |
|  |  |
|  | void addFraction::add(addFraction frac1, addFraction frac2) { |
|  | numerator = (frac1.numerator \* frac2.denominator) + (frac1.denominator \* frac2.numerator); |
|  | denominator = frac1.denominator \* frac2.denominator; |
|  | } |
|  |  |
|  | void addFraction::dispFraction() { |
|  | cout <<"\nAddition of two fraction is: " <<numerator <<"/" <<denominator; |
|  | } |
|  |  |
|  | int main() { |
|  | char choice; |
|  | addFraction fraction1, fraction2, fraction3; |
|  | do { |
|  | cout <<"\nEnter first fraction in (a/b) format: "; |
|  | fraction1.getFraction(); |
|  | cout <<"Enter second fraction in (a/b) format: "; |
|  | fraction2.getFraction(); |
|  | fraction3.add(fraction1, fraction2); |
|  | fraction3.dispFraction(); |
|  | cout <<"\n\nDo you want to continue(y,n): "; |
|  | cin >>choice; |
|  | } while(choice == 'Y' || choice == 'y'); |
|  | cout <<endl; |
|  | return 0; |
|  | } |

10. Create a class called ship that incorporates a ship’s number and location. Use the  
approach of Exercise 8 to number each ship object as it is created. Use two variables of  
the angle class from Exercise 7 to represent the ship’s latitude and longitude. A member  
function of the ship class should get a position from the user and store it in the object;  
another should report the serial number and position. Write a main() program that creates three ships, asks the user to input the position of each, and then displays each ship’s  
number and position.

Solve:

| #include <iostream> |
| --- |
|  | using namespace std; |
|  |  |
|  | class angle { |
|  | private: |
|  | int degrees; |
|  | float minutes; |
|  | char direction; |
|  | public: |
|  | void getAngle(); |
|  | void dispAngle() const; |
|  | }; |
|  |  |
|  | class ship { |
|  | private: |
|  | angle latitude, longitude; |
|  | static int count; |
|  | unsigned int shipNumber; |
|  | public: |
|  | ship() { |
|  | count++; |
|  | shipNumber = count; |
|  | } |
|  | void getPosition(); |
|  | void display() const; |
|  | }; |
|  |  |
|  | int ship::count = 0; |
|  |  |
|  | //member function defination |
|  | void angle::getAngle() { |
|  | cout <<"\nEnter degrees: "; |
|  | cin >>degrees; |
|  | cout <<"Enter minutes: "; |
|  | cin >>minutes; |
|  | cout <<"Enter directions(E, W, N, S): "; |
|  | cin >>direction; |
|  | } |
|  |  |
|  | void angle::dispAngle() const{ |
|  | cout <<degrees <<"\xF8" <<minutes <<"\' " <<direction; |
|  | if(direction == 'E' || direction == 'e' || direction == 'W' || direction == 'w') { |
|  | cout <<" Longitude !"; |
|  | } if(direction == 'N' || direction == 'n' || direction == 'S' || direction == 's') { |
|  | cout <<" Latitude !"; |
|  | } |
|  | } |
|  |  |
|  | void ship::getPosition() { |
|  | cout <<"\nEnter longitude of ship: "; |
|  | longitude.getAngle(); |
|  | cout <<"\nEnter latitude of ship: "; |
|  | latitude.getAngle(); |
|  | } |
|  |  |
|  | void ship::display() const{ |
|  | cout <<"Ship Serial Number: " <<shipNumber <<endl; |
|  | longitude.dispAngle(); |
|  | cout <<"\n"; |
|  | latitude.dispAngle(); |
|  | } |
|  |  |
|  | int main() { |
|  | ship ship1, ship2, ship3; |
|  | cout <<"\nEnter first ship position: "; |
|  | ship1.getPosition(); |
|  | cout <<"\nEnter second ship position: "; |
|  | ship2.getPosition(); |
|  | cout <<"\nEnter third ship position: "; |
|  | ship3.getPosition(); |
|  |  |
|  | cout <<"\nFirst Ship Position is: \n"; |
|  | ship1.display(); |
|  | cout <<endl; |
|  | cout <<"\nSecond Ship Position is: \n"; |
|  | ship2.display(); |
|  | cout <<endl; |
|  | cout <<"\nThird Ship Position is: \n"; |
|  | ship3.display(); |
|  | cout <<endl <<endl; |
|  |  |
|  | return 0; |
|  | } |

11. Modify the four-function fraction calculator of Exercise 12 in Chapter 5 to use a  
fraction class rather than a structure. There should be member functions for input and  
output, as well as for the four arithmetical operations. While you’re at it, you might as  
well install the capability to reduce fractions to lowest terms. Here’s a member function  
that will reduce the fraction object of which it is a member to lowest terms. It finds the  
greatest common divisor (gcd) of the fraction’s numerator and denominator, and uses this  
gcd to divide both numbers.  
void fraction::lowterms() // change ourself to lowest terms  
{  
long tnum, tden, temp, gcd;  
tnum = labs(num); // use non-negative copies  
tden = labs(den); // (needs cmath)  
if(tden==0 ) // check for n/0  
{ cout << “Illegal fraction: division by 0”; exit(1); }  
else if( tnum==0 ) // check for 0/n  
{ num=0; den = 1; return; }  
// this ‘while’ loop finds the gcd of tnum and tden  
while(tnum != 0)  
{  
if(tnum < tden) // ensure numerator larger  
{ temp=tnum; tnum=tden; tden=temp; } // swap them  
tnum = tnum - tden; // subtract them  
}  
gcd = tden; // this is greatest common divisor  
num = num / gcd; // divide both num and den by gcd  
den = den / gcd; // to reduce frac to lowest terms  
}  
You can call this function at the end of each arithmetic function, or just before you perform output. You’ll also need the usual member functions: four arithmetic operations,  
input, and display. You may find a two-argument constructor useful.

Solve:

| #include <iostream> |
| --- |
|  | #include <cstdlib> |
|  | using namespace std; |
|  |  |
|  | class fraction { |
|  | private: |
|  | int numerator; |
|  | int denominator; |
|  | public: |
|  | fraction():numerator(0), denominator(0) |
|  | { } |
|  | void getFraction(); |
|  | void arithOperation(fraction, fraction, char); |
|  | void dispFraction() const; |
|  | void lowterms(); |
|  | }; |
|  |  |
|  | /\* ----- Member function defination outside class ----- \*/ |
|  | //getFraction() |
|  | void fraction::getFraction() { |
|  | char dummychar; |
|  | cout <<"\nEnter a fraction in a/b form: "; |
|  | cin >>numerator >>dummychar >>denominator; |
|  | } |
|  |  |
|  | void fraction::arithOperation(fraction frac1, fraction frac2, char operatr) { |
|  |  |
|  | switch(operatr) { |
|  | case '+': |
|  | cout <<"\nAddition of two fraction is: "; |
|  | numerator = (frac1.numerator \* frac2.denominator) + (frac1.denominator \* frac2.numerator); |
|  | denominator = frac1.denominator \* frac2.denominator; |
|  | break; |
|  |  |
|  | case '-': |
|  | cout <<"\nSubtraction of two fraction is: "; |
|  | numerator = (frac1.numerator \* frac2.denominator) - (frac1.denominator \* frac2.numerator); |
|  | denominator = frac1.denominator \* frac2.denominator; |
|  | break; |
|  |  |
|  | case '\*': |
|  | cout <<"\nMultiplication of two fraction is: "; |
|  | numerator = frac1.numerator \* frac2.numerator; |
|  | denominator = frac1.denominator \* frac2.denominator; |
|  | break; |
|  |  |
|  | case '/': |
|  | cout <<"\nDivision of two fraction is: "; |
|  | numerator = frac1.numerator \* frac2.denominator; |
|  | denominator = frac1.denominator \* frac2.numerator; |
|  | break; |
|  |  |
|  | } |
|  | } |
|  |  |
|  | //dispFraction() |
|  | void fraction::dispFraction() const{ |
|  | cout <<numerator <<"/" <<denominator; |
|  | } |
|  |  |
|  | //lowTerms() |
|  | void fraction::lowTerms() { |
|  | long tnum, tden, temp, gcd; |
|  |  |
|  | tnum = labs(numerator); |
|  | tden = labs(denominator); |
|  |  |
|  | if(tden == 0) { |
|  | cout <<"Illegasl fraction: division by 0"; |
|  | exit(1); |
|  | } else if (tnum == 0) { |
|  | numerator = 0; |
|  | denominator = 1; |
|  | return; |
|  | } |
|  | while(tnum != 0) { |
|  | if(tnum <tden) { |
|  | temp = tnum; |
|  | tnum = tden; |
|  | tden = temp; |
|  | } |
|  | tnum = tnum - tden; |
|  | } |
|  | gcd = tden; |
|  | numerator = numerator/gcd; |
|  | denominator = denominator/gcd; |
|  | cout <<numerator <<"/" <<denominator; |
|  | cout <<endl; |
|  | } |
|  |  |
|  | int main() { |
|  | fraction fraction1, fraction2, fraction3; |
|  | char oprt; |
|  | cout <<"\nEnter first fraction: "; |
|  | fraction1.getFraction(); |
|  | cout <<"\nEnter second fraction: "; |
|  | fraction2.getFraction(); |
|  | cout <<"\nEnter an operator for arithmatic operation: "; |
|  | cin >>oprt; |
|  | fraction3.arithOperation(fraction1, fraction2, oprt); |
|  | fraction3.dispFraction(); |
|  | cout <<"\nLoswest terms of fraction is: "; |
|  | fraction3.lowTerms(); |
|  |  |
|  | cout <<endl; |
|  | return 0; |
|  | } |