ICT159 Assignment

Jin Cherng Chong
33170193

Question 1

1. Assumptions

1) Assume that the customer will input the data of the correct type. So in my program, one of the things I asked the user to input is an integer for amount. So the user will be assumed to input an Integer amount.
2) Assumes a customer will input some data (amount)
3) Assume the customer will want the change coins and not in credit card
4) Assume that the customer wants the change given with as little coins as possible
5) Assume that currency is Australian (AUD)
6) Assume the customer will want the change in coins and not in credit card
7) Assume the customer has no preference into what coin he wants. Eg: he doesn't want change in mostly 20 cent coins.
8) Assume the customer does not incur GST for his transaction
9) Bank teller has enough coins for the many customers and the many returning customers
10) Assume that the user will give the amount in form of "xx" not "0.xx" so not decimal form.

2. Algorithm

```
*****************************************************************
*************************
Author: Jin cherng chong
Date: 07/03/2018
Purpose: ICT159 Question1 Assignment
*****************************************************************
*************************

Start

     Void Function Change(Integer money, Integer chgFifty,
Integer chgTwenty, Integer chgTen, Integer chgFive)
```

```
          While(50 <= money) Then
               money -= 50
               chgFifty++
          EndWhile

          While(20 <= money) Then
               money -= 20
               chgTwenty++
          EndWhile

          While(10 <= money) Then
               money -= 10
               chgTen++
          EndWhile

          While(5 <= money) Then
               money -= 5
               chgFive++
          EndWhile

          Return

     End Function




     Void Function DspChange(Integer money, Integer dspFifty,
Integer dspTwenty, Integer dspTen, Integer dspFive)

          Output "Total money given:  Return Fifty cents:  Twenty
cents:  Ten cents:  Five cents: ", money, dspFifty, dspTwenty,
dspTen, dspFive

          Return

     End Function




     Character Function ChkMoney(Integer money)

          Character chkValid

          If(money >= 5) And (money <= 95) Then
```

```
                If(money % 5 == 0) Then
                     chkValid = 'y';
                Else
                     Output "Error: cents entered in range. NOT
multiple of 5"
                EndIf
           Else
                If(money % 5 == 0) Then
                     Output "Error: cents entered NOT in range.
Multiple of 5."
                Else
                     Output "Error: cents entered NOT in range.
NOT multiple of 5"
                EndIf
           EndIf

           Return(chkValid)

      End Function





      Integer Function GetMoney(void)

           Integer gtMoney

           Output "Please enter a valid amount: "
           Input gtMoney

           Return(gtMoney)

      End Function





      Integer Function Main()

           Integer money, fifty = 0, twenty = 0, ten = 0, five = 0
           Character validAmount

           Do
                money = GetMoney()
                validAmount = ChkMoney(money)
           While(validAmount != 'y')
```

```
          Change(money, fifty, twenty, ten, five)
          DspChange(money, fifty, twenty, ten, five)

          Return(0)

     End Function

Stop
```

3. Test Table

Test #	Test Description	Inputs	Expected Outputs	Algorithm Outputs	Program Success/Failure
1	Within lower range but only just multiple of 5	5	Given: 5 Return Five cents: 1 Ten cents: 0 Twenty cents: 0 Fifty cents: 0	Total money given: 5 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 1	Success
2	Within upper range but only just and multiple of 5 value	95	Given: 95 Return Five cents: 1 Ten cents: 0 Twenty cents: 2 Fifty cents: 1	Total money given: 95 Return Fifty cents: 1 Twenty cents: 2 Ten cents: 0 Five cents: 1	Success
3	Zero Input	0	cents entered NOT in range. Multiple of 5. Please enter a valid amount	Error: cents entered NOT in range. Multiple of 5. Please enter a valid amount:	Success
4	Within range but not multiple of 5	6	Cents entered in range. NOT multiple of 5. Please enter valid number	Error: cents entered in range. NOT multiple of 5 Please enter a valid amount:	Success
5	Within Range and Multiple of 5	45	Given: 45 Return Five cents: 1 Ten cents: 0 Twenty cents: 2 Fifty cents: 0	Total money given: 45 Return Fifty cents: 0 Twenty cents: 2 Ten cents: 0 Five cents: 1	Success

6	Negative input	-1	cents entered NOT in range. NOT multiple of 5 Please enter a valid amount:	Error: cents entered NOT in range. NOT multiple of 5 Please enter a valid amount:	Success
7	Within Range and Multiple of 5 and exactly a coin value (50)	50	Given: 50 Return Five cents: 0 Ten cents: 0 Twenty cents: 0 Fifty cents: 1	Total money given: 50 Return Fifty cents: 1 Twenty cents: 0 Ten cents: 0 Five cents: 0	Success
8	Within range and multiple of 5 and not an exact coin value	60	Given: 60 Return Five cents: 0 Ten cents: 1 Twenty cents: 0 Fifty cents: 1	Total money given: 60 Return Fifty cents: 1 Twenty cents: 0 Ten cents: 1 Five cents: 0	Success
9	Within range and multiple of 5 and exactly all coin values delegated	85	Given: 85 Return Five cents: 1 Ten cents: 1 Twenty cents: 1 Fifty cents: 1	Total money given: 85 Return Fifty cents: 1 Twenty cents: 1 Ten cents: 1 Five cents: 1	Success
Test #	**Test Description**	**Inputs**	**Expected Outputs**	**Algorithm Outputs**	**Program Success/Failure**
10	Within range and multiple of 5 and exact coin value (10)	10	Given: 10 Return Five cents: 0 Ten cents: 1 Twenty cents: 0 Fifty cents: 0	Total money given: 10 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 1 Five cents: 0	Success
11	Input of incorrect data type				

12	Null input				

4. Code

```
/******************************************************************
**************************
Author: Jin cherng chong
Date: 23/04/2018
Purpose: ICT159 Question1 Assignment
******************************************************************
*************************/

#include <stdio.h>

void Change(int money, int &chgFifty, int &chgTwenty, int
&chgTen, int &chgFive) //Added chg in front to differentiate from
dspFifty and fifty for debugging purposes. Also since prefer not
to have all variables same name (style)
{
     while(50 <= money)
     {
          money -= 50;
          chgFifty++;
     }

     while(20 <= money)
     {
          money -= 20;
          chgTwenty++;
     }

     while(10 <= money)
     {
          money -= 10;
          chgTen++;
     }

     while(5 <= money)
     {
          money -= 5;
          chgFive++;
     }
     return;

}





void DspChange(int money, int dspFifty, int dspTwenty, int
dspTen, int dspFive)
{
     printf("Total money given: %d Return Fifty cents: %d Twenty
cents: %d Ten cents: %d Five cents: %d\n", money, dspFifty,
dspTwenty, dspTen, dspFive);

     return;

}





char ChkMoney(int money)
{
     char chkValid;

     if(money >= 5 && money <= 95)
    {
        if(money % 5 == 0)
        {
            chkValid = 'y';
        }
        else
        {
            printf("Error: cents entered in range. NOT multiple
of 5\n");
        }
    }

    else
    {
        if (money % 5 == 0)
        {
            printf("Error: cents entered NOT in range. Multiple
of 5\n");
        }
        else
        {
            printf("Error: cents entered NOT in range. NOT
multiple of 5\n");
        }
     }

     return(chkValid);

}




int GetMoney(void)
{
     int gtMoney;

     printf("Please enter a valid amount: "); //Not error testing
if a user types 40.86 etc since it is assumed that the value
imputed would be an integer.
     scanf("%d%*c", &gtMoney);

     return(gtMoney);

}




int main()
{
     int money, fifty = 0, twenty = 0, ten = 0, five = 0;
     char validAmount;

     do
     {
          money = GetMoney();
          validAmount = ChkMoney(money);
     } while(validAmount != 'y');

     Change(money, fifty, twenty, ten, five);
     DspChange(money, fifty, twenty, ten, five);

     return(0);

}
```

5. Results of Program Testing

```
TestCase 1:
Please enter a valid amount: 5
Total money given: 5 Return fifty cents: 0 twenty cents: 0 ten cents: 0 five cents: 1

TestCase 2:
Please enter a valid amount: 95
Total money given: 95 Return Fifty cents: 1 Twenty cents: 2 Ten cents: 0 Five cents: 1

TestCase 3:
Please enter a valid amount: 0
Error: cents entered NOT in range. Multiple of 5
Please enter a valid amount:

TestCase 4:
Please enter a valid amount: 6
Error: cents entered in range. NOT multiple of 5
Please enter a valid amount:

TestCase 5:
Please enter a valid amount: 45
Total money given: 45 Return Fifty cents: 0 Twenty cents: 2 Ten cents: 0 Five cents: 1

TestCase 6:
Please enter a valid amount: -1
Error: cents entered NOT in range. NOT multiple of 5
Please enter a valid amount:

TestCase 7:
Please enter a valid amount: 50
Total money given: 50 Return Fifty cents: 1 Twenty cents: 0 Ten cents: 0 Five cents: 0

TestCase 8:
Please enter a valid amount: 60
Total money given: 60 Return Fifty cents: 1 Twenty cents: 0 Ten cents: 1 Five cents: 0

TestCase 9:
Please enter a valid amount: 85
Total money given: 85 Return Fifty cents: 1 Twenty cents: 1 Ten cents: 1 Five cents: 1

TestCase 10:
Please enter a valid amount: 10
Total money given: 10 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 1 Five cents: 0
```

6. Self Assessment

The first part of the assessment required us to create a c program, which allowed a customer to input an amount of money and the money would be given back in coins. This activity at first was quite perplexing. However, I had at least some sort of idea on how to do it which was all I really needed. There were four principles that this assignment needed to abide by.

The first principle that influenced my assignment was low coupling. Low coupling aims to limit the interaction with data between modules. This principle was quite difficult to abide since there were many different possible solutions to the assessment and the issue came with deciding how many modules I wanted. Originally, I planned for one of my module to call another module (not the main) but this would have brought about unnecessary interactions. Therefore, in the end I stuck to the strategy that the past workshops used and that was making all modules interact with the main function.

The second principle that influenced my assignment was high cohesion. The principle of high cohesion is where each module should solve one part of the problem. In my program, I wanted to combine the display module and calculation function module together. I thought it would look neater however; this principle stopped me from doing so. Since combing the display module and the calculation model would mean a single module solves two parts of the problem thus breaking the principle of high cohesion.

Another principle that influenced my assignment was code-reuse. This principle aims to make the solution general enough so it can more broadly useful then just to solve the immediate problem. To adhere to this principle I made sure I had my modules as generally as possible and abided by the rule of one part one module. In addition, I made sure that the first modules could be reused for the second part of the assignment. So no additional modules would be required to solve the second part. Also I wanted to make sure the first part of the assignment was very similar to the second part of the assignment. So the original variable names and code would be reused for the second part of the assignment; picking variable names that was logical and could be understood clearly was quite difficult.

Question 2

1. Assumptions

- Customer does not provide negative cents no eg: 40.-40
- Customer will not provide cents in 3 decimal places only maximum 2 decimal places
- If the customer wants cents then assume they input it in decimal form ie if they want 40 cents they will type "0.xx" → "0.40" ie
- Customer will give money in either data type of float or integer (but integers entered will also consider the other assumptions in question 2 such as it can't be negative integers and others)
- Assume that if a negative dollar is provided the cents provided will also be negative. Eg: -40.40 = -40 dollars and -40 cents
- Assume if the if a customer wants 5 cents then the 5 will be in the second decimal place ie "0.05"
- 0 is a multiple of 5

2. Algorithm

```
*****************************************************************
*************************
Author: Jin cherng chong
Date: 07/03/2018
Purpose: ICT159 Q2 Algorithm
*****************************************************************
*************************

Start

     Void Function Change(Integer cents, Integer dollars, Integer
chgFifty, Integer chgTwenty, Integer chgTen, Integer chgFive,
Integer chgHundredD, Integer chgFiftyD, Integer chgTwentyD,
Integer chgTenD, Integer chgFiveD, Integer chgTwoD, Integer
chgOneD)

          While(50 <= cents) Then
               cents -= 50
               chgFifty++
          EndWhile

          While(20 <= cents) Then
               cents -= 20
               chgTwenty++
          EndWhile

          While(10 <= cents) Then
```

```
            cents -= 10
            chgTen++
       EndWhile

       While(5 <= cents) Then
            cents -= 5
            chgFive++
       EndWhile


       While(100 <= dollars) Then
            dollars -= 100
            chgHundredD++
       EndWhile

       While(50 <= dollars) Then
            dollars -= 50
            chgFiftyD++
       EndWhile

       While(20 <= dollars) Then
            dollars -= 20
            chgTwentyD++
       EndWhile

       While(10 <= dollars) Then
            dollars -= 10
            chgTenD++
       EndWhile

       While(5 <= dollars) Then
            dollars -= 5
            chgFiveD++
       EndWhile

       While(2 <= dollars) Then
            dollars -= 2
            chgTwoD++
       EndWhile

       While(1 <= dollars) Then
            dollars -= 1
            chgOneD++
       EndWhile

       Return

  End Function
```

```
     Void Function DspChange(Integer money, Integer dspFifty,
Integer dspTwenty, Integer dspTen, Integer dspFive, Integer
dspHundredD, Integer dspFiftyD, Integer dspTwentyD, Integer
dspTenD, Integer dspFiveD, Integer dspTwoD, Integer dspOneD)

          Output "Total money given:  ", money
          Output "Return Hundred dollars:  Fifty dollars:  Twenty
dollars:  Ten dollars:  Five dollars:  Two dollars:  One dollars:
", dspHundredD, dspFiftyD, dspTwentyD, dspTenD, dspFiveD,
dspTwoD, dspOneD
          Output "Return Fifty cents:  Twenty cents:  Ten cents:
Five cents:  ",  dspFifty, dspTwenty, dspTen, dspFive
          Return;

     End Function





     Character Function ChkMoney(Float money, Integer cents,
Integer dollars)

          Character chkValid

          dollars = (int) money
          cents = (int) (((money - dollars)*100) + 0.5)

          If(cents % 5 == 0) Then
               If(dollars >= 0) Then
                    chkValid = 'y'
               Else
                    Output "Error: cents is multiple of 5.
Dollars is positive"
               EndIf
          Else
               If(dollars >= 0) Then
                    Output "Error: cents is NOT multiple of 5.
Dollar is positive"
               Else
                    Output "Error: cents is NOT multiple of 5.
Dollar is negative"
               EndIf
         EndIf
```

```
          Return(chkValid)

     End Function




     Float Function GetMoney(void)

          Float gtMoney

          Output "Please enter a valid amount: "
          Input gtMoney

          If(gtMoney < 0) Then
               Output "Warning!!! You've entered a negative
               value! Not only does it not make sense but the
               program may give you wrong error."
          EndIf

          Return(gtMoney)

     End Function




     Integer Function Main()

          Integer cents, dollars, fifty = 0, twenty = 0, ten = 0,
five = 0, hundredD = 0, fiftyD = 0, twentyD = 0, tenD = 0, fiveD
= 0, twoD = 0, oneD = 0
          Float money
          Character validAmount

          Do
               money = GetMoney()
               validAmount = ChkMoney(money, cents, dollars)
          While(validAmount != 'y')

          Change(cents, dollars, fifty, twenty, ten, five,
hundredD, fiftyD, twentyD, tenD, fiveD, twoD, oneD)
          DspChange(money, fifty, twenty, ten, five, hundredD,
fiftyD, twentyD, tenD, fiveD, twoD, oneD)

          Return(0)
```

```
        End Function

    Stop
```

3. Test Table

Test #	Test Description	Inputs	Expected Outputs	Algorithm Outputs	Program Success/Failure
1	All "valid dollar values" delegated and cents	188.85	Total money given: 188.85 Return Hundred dollars: 1 Fifty dollars: 1 Twenty dollars: 1 Ten dollars: 1 Five dollars: 1 Two dollars: 1 One dollars: 1 Fifty cents: 1 Twenty cents: 1 Ten cents: 1 Five cents: 1	Total money given: 188.85 Return Hundred dollars: 1 Fifty dollars: 1 Twenty dollars: 1 Ten dollars: 1 Five dollars: 1 Two dollars: 1 One dollars: 1 Return Fifty cents: 1 Twenty cents: 1 Ten cents: 1 Five cents: 1	Success
2	All "valid dollar values" are delegated	188	Total money given: 188.00 Return Hundred dollars: 1 Fifty dollars: 1 Twenty dollars: 1 Ten dollars: 1 Five dollars: 1 Two dollars: 1 One dollars: 1 Return Fifty cents: 0 Twenty cents: 0 Ten cents	Total money given: 188.00 Return Hundred dollars: 1 Fifty dollars: 1 Twenty dollars: 1 Ten dollars: 1 Five dollars: 1 Two dollars: 1 One dollars: 1 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0	Success
3	Negative Input	-10.50	Negative amount doesn't make sense cents is NOT multiple of 5. Dollar is negative	Warning!!! You've entered a negative value! Not only does it not make sense but the program may give you wrong error. Error: cents is NOT multiple of 5. Dollar is negative	Success
4	A large "valid dollar values" and maximum cents value	1400.95	Total money given: 1400.95 Return Hundred dollars: 14 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Fifty cents: 1 Twenty cents: 2 Ten cents: 0 Five cents: 1	Total money given: 1400.95 Return Hundred dollars: 14 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Return Fifty cents: 1 Twenty cents: 2 Ten cents: 0 Five cents: 1	Success
5	Negative Dollars (Integer)	-10	Negative amount doesn't make sense cents is multiple of 5. Dollar is negative	Warning!!! You've entered a negative value! Not only does it not make sense but the program may give you wrong error. Error: cents is multiple of 5. Dollars is positive.	Success
6	Just outside the last "valid dollar values" but a "valid dollar amount"	101	Total money given: 101.00 Return Hundred dollars: 1 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 1 Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0	Total money given: 101.00 Return Hundred dollars: 1 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 1 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0	Success

7	Enters cents as integer with no dollars	20.00	Total money give: 20.00 Return Hundred dollars: 0 Fifty dollars: 0 Twenty dollars: 1 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0	Total money given: 20.00 Return Hundred dollars: 0 Fifty dollars: 0 Twenty dollars: 1 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0	Success
8	A "valid dollar amount" but the amount is a big amount	1250	Total money give: 1250.00 Return Hundred dollars: 12 Fifty dollars: 1 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Fifty cents: 0 Twenty cents: 0 Ten cents Five cents: 0	Total money given: 1250.00 Return Hundred dollars: 12 Fifty dollars: 1 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0	Success
9	A "valid dollar value" but the cents given is not a multiple	20.96	cents is NOT multiple of 5. Dollar is positive	Error: cents is NOT multiple of 5. Dollar is positive	Success
Test #	**Test Description**	**Inputs**	**Expected Outputs**	**Algorithm Outputs**	**Program Success/Failure**
10	Enter 5 cents	0.05	Total money give: 0.05 Return Hundred dollars: 0 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 1	Total money give: 0.05 Return Hundred dollars: 0 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two dollars: 0 One dollars: 0 Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 1	Success
11	Input of incorrect data type				
12	Null input				

4. Code

```
/*****************************************************************
*************************
Author: Jin cherng chong
Date: 23/04/2018
Purpose: ICT159 Question2 Assignment
******************************************************************
*************************/

#include <stdio.h>

void Change(int cents, int dollars, int &chgFifty, int
&chgTwenty, int &chgTen, int &chgFive, int &chgHundredD, int
&chgFiftyD, int &chgTwentyD, int &chgTenD, int &chgFiveD, int
&chgTwoD, int &chgOneD)
{
     while(50 <= cents)
     {
          cents -= 50;
          chgFifty++;
     }

     while(20 <= cents)
     {
          cents -= 20;
          chgTwenty++;
     }

     while(10 <= cents)
     {
          cents -= 10;
          chgTen++;
     }

     while(5 <= cents)
     {
          cents -= 5;
          chgFive++;
     }


     while(100 <= dollars)
     {
          dollars -= 100;
          chgHundredD++;
     }

     while(50 <= dollars)
     {
          dollars -= 50;
          chgFiftyD++;
     }

     while(20 <= dollars)
     {
          dollars -= 20;
          chgTwentyD++;
     }

     while(10 <= dollars)
     {
          dollars -= 10;
          chgTenD++;
     }

     while(5 <= dollars)
     {
          dollars -= 5;
          chgFiveD++;
     }

     while(2 <= dollars)
     {
          dollars -= 2;
          chgTwoD++;
     }

     while(1 <= dollars)
     {
          dollars -= 1;
          chgOneD++;
     }

     return;

}





void DspChange(float money, int dspFifty, int dspTwenty, int
dspTen, int dspFive, int dspHundredD, int dspFiftyD, int
dspTwentyD, int dspTenD, int dspFiveD, int dspTwoD, int dspOneD)
{

     printf("Total money given: %.2f\n", money);
     printf("Return Hundred dollars: %d Fifty dollars: %d Twenty
dollars: %d Ten dollars: %d Five dollars: %d Two dollars: %d One
dollars: %d\n", dspHundredD, dspFiftyD, dspTwentyD, dspTenD,
dspFiveD, dspTwoD, dspOneD);
     printf("Return Fifty cents: %d Twenty cents: %d Ten cents:
%d Five cents: %d\n", dspFifty, dspTwenty, dspTen, dspFive);

     return;

}





char ChkMoney(float money, int &cents, int &dollars)
{
     char chkValid;

     dollars = (int) money;
    cents = (int) (((money - dollars)*100) + 0.5);

    if(cents % 5 == 0)
    {
        if(dollars >= 0)
        {
            chkValid = 'y';
        }
        else
        {
            printf("Error: cents is multiple of 5. Dollars is
positive\n");
        }
    }
    else
    {
        if (dollars >= 0)
        {
            printf("Error: cents is NOT multiple of 5. Dollar is
positive\n");
        }
        else
        {
            printf("Error: cents is NOT multiple of 5. Dollar is
negative\n");
        }
     }

     return(chkValid);
```

```
}





float GetMoney(void)
{
     float gtMoney;

     printf("Please enter a valid amount: ");
     scanf("%f%*c", &gtMoney);

     if(gtMoney < 0)
     {
          printf("Warning!!! You've entered a negative value! Not
          only does it not make sense but the program may give
          you wrong error.\n");
     }

     return(gtMoney);

}





int main()
{
     int cents = 0, dollars = 0, fifty = 0, twenty = 0, ten = 0,
five = 0, hundredD = 0, fiftyD = 0, twentyD = 0, tenD = 0, fiveD
= 0, twoD = 0, oneD = 0;
     float money;
     char validAmount;

     do
     {
          money = GetMoney();
          validAmount = ChkMoney(money, cents, dollars);
     } while(validAmount != 'y');

     Change(cents, dollars, fifty, twenty, ten, five, hundredD,
fiftyD, twentyD, tenD, fiveD, twoD, oneD);
     DspChange(money, fifty, twenty, ten, five, hundredD, fiftyD,
twentyD, tenD, fiveD, twoD, oneD);

     return(0);
```

}

5. Results of Program Testing

```
TestCase 1:
Please enter a valid amount: 188.85
Total money given: 188.85
Return Hundred dollars: 1 Fifty dollars: 1 Twenty dollars: 1 Ten dollars: 1 Five dollars: 1 Two
dollars: 1 One dollars: 1
Return Fifty cents: 1 Twenty cents: 1 Ten cents: 1 Five cents: 1



TestCase 2:
Please enter a valid amount: 188
Total money given: 188.00
Return Hundred dollars: 1 Fifty dollars: 1 Twenty dollars: 1 Ten dollars: 1 Five dollars: 1 Two
dollars: 1 One dollars: 1
Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0


TestCase 3:
Please enter a valid amount: -10.50
Warning!!! You've entered a negative value! Not only does it not make sense but the program
may give you wrong error.
Error: cents is NOT multiple of 5. Dollars is negative
Please enter a valid amount:


TestCase 4:
Please enter a valid amount: 1400.95
Total money given: 1400.95
Return Hundred dollars: 14 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two
dollars: 0 One dollars: 0
Return Fifty cents: 1 Twenty cents: 2 Ten cents: 0 Five cents: 1



TestCase 5:
Please enter a valid amount: -10
Warning!!! You've entered a negative value! Not only does it not make sense but the program
may give you wrong error.
Error: cents is multiple of 5. Dollar is positive
Please enter a valid amount:

TestCase 6:
Please enter a valid amount: 101
Total money given: 101.00
Return Hundred dollars: 1 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two
dollars: 0 One dollars: 1
```

```
Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0


TestCase 7:
Please enter a valid amount: 20.00
Total money give: 20.00
Return Hundred dollars: 0 Fifty dollars: 0 Twenty dollars: 1 Ten dollars: 0 Five dollars: 0 Two
dollars: 0 One dollars: 0
Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0

TestCase 8:
Please enter a valid amount: 1250
Total money given: 1250.00
Return Hundred dollars: 12 Fifty dollars: 1 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two
dollars: 0 One dollars: 0
Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 0


TestCase 9:
Please enter a valid amount: 20.96
Error: cents is NOT multiple of 5. Dollar is positive
Please enter a valid amount:

TestCase 10:
Please enter a valid amount: 0.05
Total money give: 0.05
Return Hundred dollars: 0 Fifty dollars: 0 Twenty dollars: 0 Ten dollars: 0 Five dollars: 0 Two
dollars: 0 One dollars: 0
Return Fifty cents: 0 Twenty cents: 0 Ten cents: 0 Five cents: 1
```

6. Self Assessment

One of the difficulties with this part of the assessment is the method of converting a dollars and cents (float) and breaking it into dollars (int) and cents (int). The method of braking down a float to cents and dollars was given to us. The result of providing the formula is that I did not quite understand how the formula worked therefore any issues I had with the formula I could not rectify.

The biggest issue was if a customer were to enter a value with three decimal then issues would arise. I tried to find methods to solve this (while still using the formula) but it was beyond my abilities and this unit's expectation. The scanf does not allow for precisions, which was one of my solutions. The biggest issue with allowing three decimal place values has to do with the inconsistency with rounding when converting. For example if a user entered "20.645" the float cents would be rounded up to "65" cents, while if the user entered "20.345" the float cents would be rounded down to "34" cents. As you can see one of the user's entries is rounded up, "64", while the other stays the same, "34". Unfortunately, the problems lies with the method that was provided to all the students to convert float to dollars and cents. The best method that I found to deal with the issue is to assume that customers will not enter a value with three decimal places. Other methods such as just allowing the varying rounding to happen is too inconsistent.

The second issue with the formula is that if a user enters a negative integer then errors will occur. For example, if a user enters “-10” then the output would state that -10 is a positive. What the program should output for -10 entry is that the dollar value is negative. However, if a user enters a negative float “-10.50” then the program works out that the dollars as being a negative. This issue is a result of the formula provided. However, as you noticed for my GetMoney() function I added error checking by stating the condition that if the money is entered is <0 then a warning message would pop up. Therefore, my method of dealing with negative integer values was to give a warning. The reasons why I didn’t implement the condition in the first part (question 1) of my assignment is because the first part of the assignment could error check and deal with negative integer values without stuffing up. Again the reason why the Question2 program can’t deal with negative values too well is because of the formula processing of the data. Alternatively, I could have made an assumption that no negative integer values were entered but I wanted my program to be more robust.

Another issues I had initially was to do with decimal places for the outputs of money given. My first version of the program outputted the total money given with 4 decimal points. So if a user enter 80 then the total money given output would be 80.0000. I fixed this by adding a "%.2f" to round two decimal places. Instead of 80.0000, it would print 80.00, which is more realistic for currency. Also adding “"%.2f" allowed for $40.4 (40 cents) to print as 40.40.

The third issue I had was trying to reduce the amount of variables I needed. My originally plan was to use global variables but the use of global variables goes against low cohesion. It would result in more modules interacting with each other. So my alternative method was using arrays but the issue with using arrays is that it is incredible difficult to pass by reference them onto another module. So in the end the only method that was easy and reduced low cohesion was in fact having multiple variables and pass by referencing them only other modules.