

CSE 1062 **Fundamentals of Programming**

Lecture #6

Spring 2016



Computer Science & Engineering Program
The School of EE & Computing
Adama Science & Technology University

- I/O statements and functions Practice
 - Case Study: Taxi Fare Change
 - Formatting Number for Program Output
 - Casting Operator
 - Case Study: Buying Pizza
 - General Problems(Maths and Physics)
 - **Assignment I: Rubber Ball**

Case Study: Taxi Fare Change



ASTU



- In Addis, to travel from Bole to Kazanchi the tariff is 2 birr and 70 cents. Suppose that the taxi assistant(conductor) don't have any birr notes but only cents to return. So he wants the change to contain as many 50 cents as possible then 25 cents, 10 cents and 5 cents, in that order. We want our program to calculate the change, given the amount paid.

Step 1: Analyze the Problem

- Input: Amount Paid
- Output: Equivalent Change in 50 cents, 25 cents, 10 cents, and 5 cents.



Step 2: Develop a Solution

- Suppose amount paid is 10 birr, therefore the change is 7 birr & 30 cents
 - Change=7.30, So cents=730
 - No of **50 cents**= $730/50=14$
 - Remaining Change= $730 \% 50=30$
 - No of **25 cents**= $30/25=1$
 - Remaining Change= $30 \% 25=5$
 - No of **10 cents**= $5/10=0$
 - Remaining Change= $5 \% 10=5$
 - No of **5 cents**= $5/5=1$

Step 2: Develop a Solution(Detailed Algorithm)



ASTU

- Prompt the user for input.
- Get input.
- Display the entered amount on the screen. (Echo)
- Calculate the Change in cents
- Print the Change in cents
- Compute and print the number of 50 cents.
- Calculate the remaining change.
- Compute and print the number of 25 cents.
- Calculate the remaining change.
- Compute and print the number of 10 cents.
- Calculate the remaining change.
- Print the remaining change(i.e. number of 5 cents)

- To calculate the equivalent change, the program performs calculations using the values of a fifty-cent, which is 50; a twenty-five-cent, which is 25; a ten-cent, which is 10 .
 - Because these data are special and the program uses these values more than once, it makes sense to declare them as named constants.
 - Using named constants also simplifies later modification of the program

1. Write a C++ program to solve the problem

- Use the following constants in your program

```
1  #include <iostream>
2  using namespace std;
3  //Named constants
4  const int FIFTY = 50;
5  const int TWENTYFIVE = 25;
6  const int TEN = 10;
7
```

- Use variables where necessary

2. Modify the program to work for payments made for two or more people

- The field width manipulator must be included for each value in the data stream sent to **cout**
- Other manipulators remain in effect until they are changed
- **iomanip** header file must be included to use manipulators requiring arguments

```
cout<<setw(5)<<fixed<<setprecision(1)<<3.23;  
cout<<setw(4)<<5.33;
```

- Test the format manipulators using a C++ program

Manipulators	Number	Display	Comments
<code>setw(2)</code>	3	3	Number fits in the field.
<code>setw(2)</code>	43	43	Number fits in the field.
<code>setw(2)</code>	143	143	Field width is ignored.
<code>setw(2)</code>	2.3	2.3	Field width is ignored.
<code>setw(5)</code> <code>fixed</code> <code>setprecision(2)</code>	2.366	2.37	Field width of five with two decimal digits.

Formatting Number for Program Output



ASTU

- Test the format manipulators using a C++ program

```
cout << "|" << setw(10) << fixed  
      << setprecision(3) << 25.67 << "|";
```

Manipulators	Number	Display	Comments
setw(5) fixed setprecision(2)	42.3	42.30	Number fits in the field with the specified precision. Note that the decimal point takes up one location in the field width.
setw(5) setprecision(2)	142.364	1.4e+002	Field width is ignored, and scientific notation is used with the setprecision manipulator.
setw(5) fixed setprecision(2)	142.364	142.36	Field width is ignored, but precision specification is used. The setprecision manipulator specifies the number of fractional digits.
setw(5) fixed setprecision(2)	142.366	142.37	Field width is ignored, but precision specification used. The setprecision manipulator specifies the number of fractional digits. (Note the rounding of the last decimal digit.)
setw(5) fixed setprecision(2)	142	142	Field width is used; fixed and setprecision manipulators are irrelevant because the number is an integer that specifies the total number of significant digits (integer plus fractional digits)

- **setiosflags** manipulator: Allows additional formatting:
 - Right or left justification
 - Fixed display with 6 decimal places
 - Scientific notation with exponential display
 - Display of a leading + sign
- **Parameterized manipulator:** One which requires arguments, or parameters

Formatting Number for Program Output



Flag	Meaning
<code>ios::fixed</code>	Always show the decimal point with six digits after the decimal point. Fill with trailing zeros after the decimal point, if necessary. This flag takes precedence if it's set with the <code>ios::showpoint</code> flag.
<code>ios::scientific</code>	Use exponential display in the output.
<code>ios::showpoint</code>	Always display a decimal point and six significant digits total (combination of integer and fractional parts). Fill with trailing zeros after the decimal point, if necessary. For larger integer values, revert to scientific notation unless the <code>ios::fixed</code> flag is set.
<code>ios::showpos</code>	Display a leading + sign when the number is positive.
<code>ios::left</code>	Left-justify the output.
<code>ios::right</code>	Right-justify the output.

```
cout << "|" << setw(10)
      << setiosflags ios::left)
      << 142 << "|";
```

- Manipulators can also be set using the **ostream** class methods
- Separate the **cout** object name from the method name with a period

Example:

```
cout.precision(2)
```

Formatting Number for Program Output



ASTU

Method	Comment	Example
<code>precision(n)</code>	Equivalent to <code>setprecision()</code>	<code>cout.precision(2)</code>
<code>fill('x')</code>	Equivalent to <code>setfill()</code>	<code>cout.fill('*')</code>
<code>setf(ios::fixed)</code>	Equivalent to <code>cout.setf(ios::fixed)</code>	<code>setiosflags(ios::fixed)</code>
<code>setf(ios::showpoint)</code>	Equivalent to <code>cout.setf(ios::showpoint)</code>	<code>setiosflags(ios::showpoint)</code>
<code>setf(ios::left)</code>	Equivalent to <code>left</code>	<code>cout.setf(ios::left)</code>
<code>setf(ios::right)</code>	Equivalent to <code>right</code>	<code>cout.setf(ios::right)</code>
<code>setf(ios::flush)</code>	Equivalent to <code>endl</code>	<code>cout.setf(ios::flush)</code>



Cast Operator

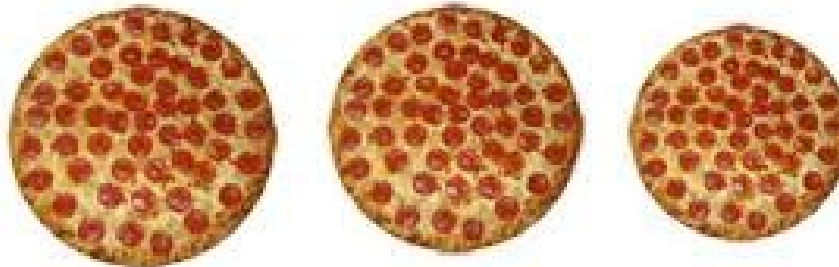
- **Cast operator:** A unary operator that forces the data to the desired data type
- Compile-time cast
 - Syntax: **dataType (expression)**
 - Example: **int(a+b)**

Cast Operator

- **Run-time cast:** The requested conversion is checked at run time and applied if valid
 - Syntax:
staticCast<data-type> (expression)
 - Example:
staticCast<int>(a*b)

- The large “economy” size of an item is not always a better buy than the smaller size.
 - This is particularly true when buying pizzas.
 - Pizza sizes are given as the diameter of the pizza in inches.

Choose Between the 14" Large, 12" Medium,
or 10" Small



- However, the quantity of pizza is determined by the area of the pizza
 - Most people cannot easily estimate the difference in area between a ten-inch pizza and a twelve-inch pizza
 - So cannot easily determine which size is the best buy—that is, which size has the lowest price per square inch.

- Build, Run and Test the following program

```
1 //Determines which of two pizza sizes is the best buy.
2 #include <iostream>
3 using namespace std;
4 void getData( int & smallDiameter, double& priceSmall,
5             int & largeDiameter, double& priceLarge);
6 void giveResults(int  smallDiameter,  double priceSmall,
7                 int  largeDiameter,  double priceLarge);
8 double unitPrice(int  diameter, double price);
9
10 int  main()
11 {
12     int diameterSmall, diameterLarge;
13     double priceSmall, priceLarge;
14     getData(diameterSmall, priceSmall, diameterLarge, priceLarge);
15     giveResults(diameterSmall, priceSmall, diameterLarge, priceLarge);
16     return 0;
17 }
```

Case Study: Buying Pizza

```
18 void getData( int & smallDiameter, double& priceSmall,
19              int & largeDiameter, double& priceLarge)
20 {
21     cout << "Welcome to the Pizza Consumers Union.\n";
22     cout << "Enter diameter of a small pizza (in inches): ";
23     cin >> smallDiameter;
24     cout << "Enter the price of a small pizza: $";
25     cin >> priceSmall;
26     cout << "Enter diameter of a large pizza (in inches): ";
27     cin >> largeDiameter;
28     cout << "Enter the price of a large pizza: $";
29     cin >> priceLarge;
30 }
```

Case Study: Buying Pizza



```
31 void giveResults(int    smallDiameter,    double priceSmall,
32                 int    largeDiameter,    double priceLarge)
33 {
34     double unitPriceSmall, unitPriceLarge;
35     unitPriceSmall = unitPrice(smallDiameter, priceSmall);
36     unitPriceLarge = unitPrice(largeDiameter, priceLarge);
37     cout.setf(ios::fixed);
38     cout.setf(ios::showpoint);
39     cout.precision(2);
40     cout << "Small pizza:\n"
41           << "Diameter = " << smallDiameter << " inches\n"
42           << "Price = $" << priceSmall
43           << " Per square inch = $" << unitPriceSmall << endl
44           << "Large pizza:\n"
45           << "Diameter = " << largeDiameter << " inches\n"
46           << "Price = $" << priceLarge
47           << " Per square inch = $" << unitPriceLarge << endl;
48     if (unitPriceLarge < unitPriceSmall)
49         cout << "The large one is the better buy.\n";
50     else
51         cout << "The small one is the better buy.\n";
52     cout << "Buon Appetito!\n";
53 }
```

Case Study: Buying Pizza

```
54 double unitPrice(int diameter, double price)
55 {
56     const double PI = 3.14159;
57     double radius, area;
58     radius = diameter/ static_cast< double>(2);
59     area = PI * radius * radius;
60     return (price/area);
61 }
62
```

- Suppose that restaurants offer both round pizzas and rectangular pizzas.



- The code is on the next slides
- Note that the function name `unitPrice` has been **overloaded** so that it applies to both round and rectangular pizzas.
- Build, Run and Test the Program

Case Study: Buying Pizza



```
1  //Determines whether a round pizza or a rectangular pizza
2  //is the best buy
3  #include <iostream>
4  using namespace std;
5  double unitPrice(int diameter, double price);
6  double unitPrice(int length, int width, double price);
7  int main( )
8  {
9      int diameter, length, width;
10     double priceRound, unitPriceRound,
11           priceRectangular, unitPriceRectangular;
12     cout << "Welcome to the Pizza Consumers Union.\n";
13     cout << "Enter the diameter in inches"
14           << " of a round pizza: ";
15     cin >> diameter;
16     cout << "Enter the price of a round pizza: $";
17     cin >> priceRound;
18     cout << "Enter length and width in inches\n"
19           << "of a rectangular pizza: ";
20     cin >> length >> width;
21     cout << "Enter the price of a rectangular pizza: $";
22     cin >> priceRectangular;
23     unitPriceRectangular = unitPrice(length, width, priceRectangular);
24     unitPriceRound = unitPrice(diameter, priceRound);
25     cout.setf(ios::fixed);
```

Case Study: Buying Pizza

```
25     cout.setf(ios::fixed);
26     cout.setf(ios::showpoint);
27     cout.precision(2);
28     cout << endl
29         << "Round pizza: Diameter = "
30         << diameter << " inches\n"
31         << "Price = $" << priceRound
32         << " Per square inch = $" << unitPriceRound
33         << endl
34         << "Rectangular pizza: Length = "
35         << length << " inches\n"
36         << "Rectangular pizza: Width = "
37         << width << " inches\n"
38         << "Price = $" << priceRectangular
39         << " Per square inch = $" << unitPriceRectangular
40         << endl;
41     if (unitPriceRound < unitPriceRectangular)
42         cout << "The round one is the better buy.\n";
43     else
44         cout << "The rectangular one is the better buy.\n";
45     cout << "Buon Appetito!\n";
46     return 0;
47 }
```

Case Study: Buying Pizza

```
48 double unitPrice(int diameter, double price)
49 {
50     const double PI = 3.14159;
51     double radius, area;
52     radius = diameter/double(2);
53     area = PI * radius * radius;
54     return (price/area);
55 }
56 double unitPrice(int length, int width, double price)
57 {
58     double area = length * width;
59     return (price/area);
60 }
61
```

- Develop a calculator program using five functions: add (), multiply (), divide (), subtract () and modulus ().

- Write a C++ program to calculate and display the value of the slope of the line connecting two points with the coordinates $(3,7)$ and $(8,12)$.
- After verifying the output your program produces, modify it to determine the slope of the line connecting the points $(2,10)$ and $(12,6)$.

- What will happen if you use the points (2,3) and (2,4)
- Change its output to this:
 - The value of the slope is xxx.xx
 - The xxx.xx denotes placing the calculated value in a field wide enough for three places to the left of the decimal point and two places to the right of it.

Assignment #1

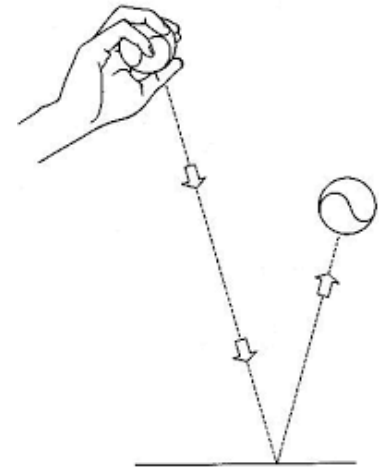


ASTU

- When a particular rubber ball is dropped from a given height (in meters), its impact speed (in meters/second) when it hits the ground is given by this formula:

$$speed = \sqrt{2gh}$$

- where g is the acceleration caused by gravity and h is the height.
- The ball then rebounds to $2/3$ the height from which it last fell.



- Write, test, and run a C++ program that calculates and displays the impact speed of the first three bounces and the rebound height of each bounce.
- Test your program by using an initial height of 2.0 meters.
- Run the program twice, and compare the results for dropping the ball on Earth ($g = 9.81 \text{ m/s}^2$) and on the moon ($g = 1.67 \text{ m/s}^2$).