POINTS: 25 if submitted on time and correctly.

Assignment: coins: convert ALL the player's money into the minimum

number of US Coins

Write a Console Application which:

asks the user for a sum of money

inputs the user's response

expresses the sum in US Coins: 25 cents, ten cents, five cents, and one cent.

doesn't make any rounding errors.

(Example: 0 = 8 % 2 or 2 = 17 % 5

% is the "modulus" operator, the remainder after dividing one number

by another.

Some get bad scores on this because they don't test their

work. That means they aren't thinking well or deeply,

and they need to know that now so they can correct it

before University. This is a different period in life:

People can die when any engineer screws up, but when you

screw up with money, people get really mad.

============================

Principle:

C DOES NOT ROUND: IT TRUNCATES;

int i = 7.9999999999 ;

There is now a 7 in i. See?

If you want rounding, remember to add 0.5 to the result of any math

operation before assigning the value to the variable.

int x = (i \* 2) + 0.5 ;

If you use "real" variable types (floats, doubles, etc.) in your

calculations, you're likely to produce wrong answers. The way we avoid

this in financial programming, is to store all dollar amounts as Pennies,

or even as Mills( tenths of a cent) , and only convert to dollars.cents

after all the calculating is done.

This, obviously, is a problem in division and subtraction.

If I have US$365.25, the number of Quarters in that sum is equal to

365.25 divided by 0.25, right?

If I had $121.78, I would first divide 121.78 by 0.25, getting

487.12. Since 12/100's of a quarter has no meaning, I'd

write:

487 quarters

Now, I want to know how much cash would be left after the quarters were

removed, so I'd subtract the amount of money the quarters represent from

the total:

121.78 - (487 x .25)

121.78 - 121.75 == 0.03

Since no dimes can go into $0.03, I'd write:

0 dimes

Also, since no Nickels can go into $0.03, I'd write:

0 nickels.

And finally, I'd write:

3 pennies.

Now I've accounted for all the money using the biggest coins first,

so 487 Quarters and 3 Pennies is the right answer.

Some of you have gotten out of practice at this Fourth Grade arithmetic,

so take out a sheet of paper and solve this problem with a pencil for

many different totals until you remember.

Asst1 Solution discussion:

You always should TEST YOUR CODE. Don't just assume you're right, you'll

be wrong most of the time. ALWAYS CHECK YOUR WORK.

If you check your work, then the first thing you want to do is verify

that the number the user entered was being properly stored, so after

you entered the number and converted it to an integral total of pennies,

PRINT IT OUT:

double input, total;

int quarters, dimes, nickels, pennies;

cout << "\nEnter a dollar amount: $ ";

cin >> input;;

total = (int) (input \* 100);

printf("total: %f\n", total) ; //PRINT IT OUT!!

Right here, you'll see if you've got an error.

Did the user enter 1.68, yet you're storing

1.67? If so, you hvae a truncation error.

If so, there are various ways you can fix it.

One of the most obvious ways is to round the

floating-point data before you copy it to an

int, since you know (from reading the book) that

NOTHING IS EVER ROUNDED, IT IS ONLY TRUNCATED. So

handle the rounding yourself:

One way:

float input, total;

int quarters, dimes, nickels, pennies;

cout << "\nEnter a dollar amount: $ ";

cin >> input;;

input += 0.005 ; // HANDLE THE ROUNDING!!

total = (int) (input \* 100);

printf("total: %f\n", total) ;

If this isn't the problem, then, are you basing your program on

multiplication (division) instead of subtraction?

As you know from 181 and 186, all division produces REALS, and REALS

are prone to rounding (truncation) errors. We learn this in Sixth Grade

when we are taught the difference between fractions, which preserve the

evidence(2/3), and decimals (which may never conclude, and can only be

stored as approximations (.6667).

While arithmetic is taught in Platonic Heaven where everything works as it

should, programming is concerned with real computers, so if you haven't

taken enough computer science, pick up some books to fill in the gaps.

Just don't expect computers to work like algebra. Algebra runs on the

human brain, our stuff runs on 64 bits, tops.

Now that I've got your attention, here are three different ways to deal

with users who want to input Reals:

/\*-----------------------------------------------------------------

\* coins0.cpp - simplest way to solve asst

\*

-----------------------------------------------------------------\*/

#include <cstdio>

#include <iostream>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

double sum ;

int q, d, n, p , total ;

cout << "Total: " ;

cin >> sum ;

sum \*= 100 ;

total = sum + 0.5 ; // not really needed, but if you're nervous

q = total / 25 ;

total -= (q \* 25 ) ;

d = total / 10 ;

total -= (d \* 10) ;

n = total / 5 ;

total -= (n \* 5 ) ;

p = total ;

printf("q: %d d: %d n: %d p: %d\n", q, d, n, p) ;

}

======================================

/\*-----------------------------------------------------------------

\* coins.cpp - parse a cash total into coins

This asst works basic arithmetic and logic, and awareness of

the importance of data type selection for ensuring accuracy

AND OF PRINTING OUT DIAGNOSTICS TO CHECK YOURSELF.

Example value: simple handling of string data, and conversion

to CString so we can use atoi().

Wed Jan 19 14:17:40 PST 2011

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cstdlib>

#include <iostream>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

int q, d, n, p, total;

string s ;

if (argc == 1) // handle command-line argument

{

cout << "Enter an amount in US currency: " ;

/\*---------

Don't use doubles, since there is a tendency to

lose pennies in rounding. Treat all money as

a sum total of integtal pennies when calculating.

---------\*/

cin >> s ;

}

else

s = argv[1] ;

// clean the string so it's # of pennies:

for (int i = 0 ; i < s.length() ; i++)

if (!isdigit(s[i]) )

{

s.erase(i,1) ;

i-- ;

}

total = atoi(s.c\_str()) ;

cout << "total: " << total << endl;

q = total / 25 ; total = total - ( q \* 25) ;

cout << "total: " << total << endl ;

d = total / 10 ; total = total - ( d \* 10) ;

cout << "total: " << total << endl ;

n = total / 5 ; total = total - (n \* 5) ;

cout << "total: " << total << endl ;

p = total ;

printf("q: %d d: %d n: %d p: %d\n", q,d,n,p) ;

} // main ends

================================================

/\*-----------------------------------------------------------------

\* coins2.cpp - parse a cash total into coins

This one is just complicated for its own sake, but it'll

be good for you to read and figure out.

Sat Feb 23 14:11:05 PST 2013

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cstdlib>

#include <iostream>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

int total, divs[] = { 25, 10, 5, 1} , ans[4], i ;

char cv[4] {'\n'} ;

string s ;

if (argc == 1)

{

cout << "Enter an amount in US currency: " ;

cin >> s ;

}

else

s = argv[1] ;

for (i = 0 ; i < s.length() ; i++)

if (!isdigit(s[i]))

s.erase(i--,1) ; // see it?

total = atoi(s.c\_str()) ;

for (i = 0 ; i < 4 ; ++i)

{

ans[i] = total / divs[i] ;

total -= divs[i] \* ans[i] ;

}

for (i = 0 ; i < 4 ; ++i)

printf("%ds: %d ", divs[i], ans[i]) ;

fflush(stdout) ;

write(1,cv,4) ; // duhwha?

} // main ends