FLAT WASHER SOLUTION

#include <stdio.h>

#include <math.h>

#define PI 3.14159 /\* Omit if PI defined in <math.h> \*/

/\* function prototypes \*/

double find\_area(double r);

double find\_rim\_area(double outer, double inner);

double find\_unit\_weight(double area, double thickness, double density);

void instruct(void);

int main(void)

{

double hole\_diameter; /\* input diameter of hole \*/

double edge\_diameter; /\* input diameter of outer edge \*/

double thickness; /\* input thickness of washer \*/

double density; /\* input density of material used \*/

double quantity; /\* input number of washers made \*/

double weight; /\* output weight of washer batch \*/

double hole\_radius; /\* radius of hole \*/

double edge\_radius; /\* radius of outer edge \*/

double rim\_area; /\* area of rim \*/

double unit\_weight; /\* weight of 1 washer \*/

/\* Give the user instructions. \*/

instruct();

/\* Get the inner diameter, outer diameter, and thickness. \*/

printf("Inner diameter in centimeters> ");

scanf("%lf", &hole\_diameter);

printf("Outer diameter in centimeters> ");

scanf("%lf", &edge\_diameter);

printf("Thickness in centimeters> ");

scanf("%lf", &thickness);

/\* Get the material density and quantity manufactured. \*/

printf("Material density in grams per cubic centimeter> ");

scanf("%lf", &density);

printf("Quantity in batch> ");

scanf("%lf", &quantity);

/\* Compute the rim area. \*/

hole\_radius = hole\_diameter / 2.0;

edge\_radius = edge\_diameter / 2.0;

rim\_area = find\_rim\_area(edge\_radius, hole\_radius);

/\* Compute the weight of a flat washer. \*/

unit\_weight = find\_unit\_weight(rim\_area, thickness, density);

/\* Compute the weight of the batch of washers. \*/

weight = unit\_weight \* quantity;

/\* Display the weight of the batch of washers. \*/

printf("\nThe expected weight of the batch is %.2f ", weight);

printf("grams.\n");

return (0);

}

/\*

\* Displays instructions to a user of program to compute the

\* weight of a batch of flat washers.

\*/

void instruct(void)

{

printf("This program computes the weight of a batch of flat \n");

printf("washers.\n\n");

printf("To use this program, please enter the inner diameter,\n");

printf("outer diameter, thickness, density, and quantity at each\n");

printf("respective prompt.\n\nThanks for using this program.\n\n");

}

/\*

\* Computes the area of a circle with radius r.

\* Pre: r is defined and is > 0.

\* PI is a constant macro representing an approximation of pi.

\* Library math.h is included.

\*/

double find\_area(double r)

{

return (PI \* pow(r, 2));

}

/\*

\* Computes the area of an annular ring with inner radius of inner

\* and outer radius of outer.

\* Pre: inner and outer are defined and are > 0.

\* Function find\_area() is defined.

\*/

double find\_rim\_area(double outer, double inner)

{

return (find\_area(outer)-find\_area(inner));

}

/\* Computes the unit weight of a flat object with an area of area,

\* with a thickness of thickness, and with a density of density.

\* Pre: area, thickness and density are defined and are > 0.

\*/

double find\_unit\_weight(double area, double thickness, double density)

{

return (area \* thickness \* density);

}

DISTANCE SOLUTION

#include <stdio.h>

/\* function prototype \*/

double find\_departure\_time (int arr\_time, double distance, double speed);

int main(void)

{

int arr\_time; /\* input--arrival time \*/

double distance, /\* input--distance traveled (km) \*/

avg\_speed; /\* input--anticipated average speed (km/hr) \*/

int dep\_time; /\* output--required departure time \*/

/\* Get arrival time \*/

printf("Enter arrival time as integer on a 24 hour clock. For example,");

printf("\n8:30 PM would be entered as 2030\n");

printf("Arrival time> ");

scanf("%d", &arr\_time);

/\* Now get the distance to travel and anticipated average speed \*/

printf("Enter the distance in km> ");

scanf("%lf", &distance);

printf("Enter anticipated average speed (including stops) in km/hr> ");

scanf("%lf", &avg\_speed);

/\* Compute and display the required departure time. \*/

dep\_time = find\_departure\_time(arr\_time, distance, avg\_speed);

printf("You need to leave at %d.\n", dep\_time );

return (0);

}

double find\_departure\_time (int arr\_time, double distance, double speed)

{

double time; /\* travel time in hours \*/

int tot\_min, /\* travel time in minutes (rounded) \*/

arr\_min, /\* total minutes to arrival time \*/

dep\_tot\_min, /\* departure time in minutes \*/

dep\_hr, /\* hour of departure (24-hr clock) \*/

dep\_min, /\* minutes of departure time \*/

dep\_time; /\* departure time (24-hr clock) \*/

time = distance / speed;

tot\_min = (int)(time \* 60 + 0.5);

arr\_min = arr\_time / 100 \* 60 + arr\_time % 100;

dep\_tot\_min = arr\_min - tot\_min;

dep\_hr = dep\_tot\_min / 60;

dep\_min = dep\_tot\_min % 60;

dep\_time = dep\_hr \* 100 + dep\_min;

return (dep\_time);

}

BISECTION SOLUTION

#include <stdio.h>

#include <math.h>

#define FALSE 0

#define TRUE 1

#define NO\_ROOT -99999.0

double bisect(double x\_left, double x\_right, double epsilon, double f(double farg));

double g(double x);

double h(double x);

int main(void)

{

double x\_left, x\_right, /\* left, right endpoints of interval \*/

epsilon, /\* error tolerance \*/

root;

/\* Get endpoints and error tolerance from user \*/

printf("\nEnter interval endpoints> ");

scanf("%lf%lf", &x\_left, &x\_right);

printf("\nEnter tolerance> ");

scanf("%lf", &epsilon);

/\* Use bisect function to look for roots of g and h \*/

printf("\n\nFunction g");

root = bisect(x\_left, x\_right, epsilon, g);

if (root != NO\_ROOT)

printf("\n g(%.7f) = %e\n", root, g(root));

printf("\n\nFunction h");

root = bisect(x\_left, x\_right, epsilon, h);

if (root != NO\_ROOT)

printf("\n h(%.7f) = %e\n", root, h(root));

return (0);

}

/\*

\* Implements the bisection method for finding a root of a function f.

\* Returns a root if signs of fp(x\_left) and fp(x\_right) are different.

\* Otherwise returns NO\_ROOT.

\*/

double bisect(double x\_left, /\* input - endpoints of interval in \*/

double x\_right, /\* which to look for a root \*/

double epsilon, /\* input - error tolerance \*/

double f(double farg)) /\* input - the function \*/

{

double x\_mid, /\* midpoint of interval \*/

f\_left, /\* f(x\_left) \*/

f\_mid, /\* f(x\_mid) \*/

f\_right; /\* f(x\_right) \*/

int root\_found; /\* flag to indicate whether root is found \*/

/\* Computes function values at initial endpoints of interval \*/

f\_left = f(x\_left); f\_right = f(x\_right);

/\* If no change of sign occurs on the interval there is not a

unique root. Exit function and return NO\_ROOT \*/

if (f\_left \* f\_right > 0) { /\* same sign \*/

printf("\nMay be no root in [%.7f, %.7f]", x\_left, x\_right);

return NO\_ROOT;

}

/\* Searches as long as interval size is large enough

and no root has been found \*/

root\_found = FALSE; /\* no root found yet \*/

while (fabs(x\_right - x\_left) > epsilon && !root\_found)

{

/\* Computes midpoint and function value at midpoint \*/

x\_mid = (x\_left + x\_right) / 2.0;

f\_mid = f(x\_mid);

if (f\_mid == 0.0) { /\* Here's the root \*/

root\_found = TRUE;

} else if (f\_left \* f\_mid < 0.0) {/\* Root in [x\_left,x\_mid]\*/

x\_right = x\_mid;

} else { /\* Root in [x\_mid,x\_right]\*/

x\_left = x\_mid;

}

/\* Trace loop execution - print root location or new interval \*/

if (root\_found)

printf("\nRoot found at x = %.7f, midpoint of [%.7f, %.7f]",

x\_mid, x\_left, x\_right);

else

printf("\nNew interval is [%.7f, %.7f]",

x\_left, x\_right);

}

/\* If there is a root, it is the midpoint of [x\_left, x\_right] \*/

return ((x\_left + x\_right) / 2.0);

}

/\* Functions for which roots are sought \*/

/\* 3 2

\* 5x - 2x + 3

\*/

double

g(double x)

{

return (5 \* pow(x, 3.0) - 2 \* pow(x, 2.0) + 3);

}

/\* 4 2

\* x - 3x - 8

\*/

double

h(double x)

{

return (pow(x, 4.0) - 3 \* pow(x, 2.0) - 8);

}

RECURSION POWER SOLUTION

int power\_raiser(int base, int power)

{

int ans;

if (power == 0)

ans = 1;

else

ans = base \* power\_raiser(base, power - 1);

return (ans);

}

RECURSION PRIME SOLUTION

1. #include <stdio.h>
3. int primeno(int, int);
5. int main()
6. {
7. int num, check;
8. printf("Enter a number: ");
9. scanf("%d", &num);
10. check = primeno(num, num / 2);
11. if (check == 1)
12. {
13. printf("%d is a prime number**\n**", num);
14. }
15. else
16. {
17. printf("%d is not a prime number**\n**", num);
18. }
19. return 0;
20. }
22. int primeno(int num, int i)
23. {
24. if (i == 1)
25. {
26. return 1;
27. }
28. else
29. {
30. if (num % i == 0)
31. {
32. return 0;
33. }
34. else
35. {
36. return primeno(num, i - 1);
37. }
38. }
39. }