ISA 221 - C
Section C - Team 4
Anoushka Das, Jaeden Patterson, Mridul Jain
Billing System

```
#-------------------------------------------------------------------------------
# Author: Anoushka Das, Jaeden Patterson, Mridul Jain
# Program: Program 5
#
# Description:
# Calculates the total amount due to each employee from the hours they worked.
# Program first calculates total hours worked, then proceeds to calculate
# overtime hours if needed. Then, calculates the billable amount of regular and
# overtime including overtime rate. Program has validation rules: name must be input, hourly rate must
# be more than $20.00, weekly hours must be within 35 and 80 hours. Allows user to input multiple
# employees.
#-------------------------------------------------------------------------------

def main():
    import BillingModule2


    #Named Constant
    max_regular_hours = 160
    overtime_rate_percentage = 1.05
    to_average = 4


    newEmployee = "y"

    while newEmployee == "y":


        #Variables
        hourlyRate = 0.00
        week = 0
        Week1 = 0
        Week2 = 0
        Week3 = 0
        Week4 = 0
        totalHours = 0.00
        overTime = 0.00
        overRate = 0.00
        extraDue = 0.00
        averageHours = 0.00
        regularDueExtra = 0.00
        regularDue = 0.00


        #Input
        employeeNameInput = "Employee Name: "
```

```
employeeName = BillingModule2.readEmployeeName(employeeNameInput)


hourlyRateInput = "Hourly Rate: "
hourlyRate = BillingModule2.readHourlyRate(hourlyRateInput)



weekInput = "Enter hours worked for week 1: "
week1 = BillingModule2.readWeeklyHours(weekInput)

weekInput = "Enter hours worked for week 2: "
week2 = BillingModule2.readWeeklyHours(weekInput)

weekInput = "Enter hours worked for week 3: "
week3 = BillingModule2.readWeeklyHours(weekInput)

weekInput = "Enter hours worked for week 4: "
week4 = BillingModule2.readWeeklyHours(weekInput)


BillingModule2.writeBillingRecord(employeeName, hourlyRate, week1, week2, week3, week4)


#Calculate Total Hours worked
totalHours = week1 + week2 + week3 + week4

#Calculate Average Hours worked
averageHours = round(totalHours/ to_average, 2)

#Calculate the invoice amount and overtime hours if applicable
if totalHours > max_regular_hours:
    #Calculate how much overtime hours worked
    overTime = round(totalHours - max_regular_hours, 2)
    #Calculate amount of overtime pay
    overRate = round(hourlyRate * overtime_rate_percentage, 2)
    extraDue = round(overTime * overRate, 2)
    #Calculate the invoice amount
    regularDue = round(max_regular_hours * hourlyRate, 2)
    invoiceAmount = regularDue + extraDue
    #Assign variables for later output printing
    workedOvertime = str(overTime) + " hours of"
    overtimeOutput = "Overtime hours: " + format(overTime, ".2f") +\
                     " @ $" + str(overRate) + " = $" +\
                     format (extraDue, ",.2f") + "\n"
```

```
            regHoursOutput = max_regular_hours

        else:
            #Calculate invoice amount without overtime
            invoiceAmount = round(totalHours * hourlyRate, 2)
            workedOvertime = "no"
            #Assign variables for later output printing
            regularDue = invoiceAmount
            overtimeOutput = ""
            #When total hours are less than 160, regular hours equals
            regHoursOutput = totalHours

        #Output

        #Print the invoice for the employee
        print("\n"+employeeName, "worked",workedOvertime, "overtime.")
        print("\nInvoice")
        print("Resource:  ", employeeName, "\tAverage weekly hours:", format(averageHours, ",.2f"))
        print("\nTotal billable hours: ", format(totalHours, ",.2f"), "\trate: $", format(hourlyRate,",.2f"),
sep="")
        print(overtimeOutput, "Regular Hours: ",\
              format(regHoursOutput, ".2f"), " @ $",\
              format(hourlyRate,".2f")," = $",\
              format(regularDue,",.2f"), sep="")
        print("Amount Due: $", format(invoiceAmount,",.2f"), sep="")


        newEmployee = input('\nEnter another employee? ("y"=yes): ')



#-------------------------------------------------------------------------------
# Author: Anoushka Das, Jaeden Patterson, Mridul Jain
# Program: BillingModule2
#
# Description:
# The given code provides a set of functions to manage billing records for employees. It includes
# functions to write billing records to a file, reset the file, and validate user input for employee
# names, hourly rates, and weekly hours worked. Additionally, there is a function to calculate the
# total billable amount due to an employee based on their hourly rate and total hours worked
#-------------------------------------------------------------------------------

def writeBillingRecord(employeeName, hourlyRate, week1, week2, week3, week4):
    outfile= open("Billing.txt", "a")
```

```
    outfile.write(employeeName + "\n"
                  + str(hourlyRate)+ "\n"
                  + str(week1)+ "\n"
                  + str(week2)+ "\n"
                  + str(week3)+ "\n"
                  + str(week4)+ "\n")

    outfile.close()



def resetBillingFile(File):
    outfile = open(File,"w").close()



def readEmployeeName(employeeNameInput):
    again = True
    while again:
            employeeName = (input(employeeNameInput))
            if employeeName == "":
                print("Employee name must be entered\n")
            else:
                again = False

    return employeeName


def readHourlyRate(hourlyRateInput):
    minHours = 20
    again = True
    while again:
        try:
            hourlyRate = float(input(hourlyRateInput))
            if hourlyRate < minHours:
                print("Invalid Hourly Rate, must be at least $20.00/hour.\n")
            else:
                again = False
        except ValueError:
            print("Hourly rate must be numeric.\n")

    return hourlyRate
```

```
def readWeeklyHours(weekInput):
    minHours = 35
    maxHours = 80
    again = True
    while again:
        try:
            week = float(input(weekInput))
            if week > maxHours or week < minHours:
                print("Invalid number of hours, must be between 35 and 80.\n")
            else:
                again = False
        except ValueError:
            print("Weekly hours worked must be numeric.\n")

    return week

def totalBillableDue (rate, totalHours):
    max_regular_hours = 160
    if totalHours > max_regular_hours:
        overtimeRate = round(rate * 1.05, 2)
        overtimeHours = totalHours - max_regular_hours
        overtimePay = round(overtimeHours * overtimeRate, 2)
        regularPay = max_regular_hours * rate
        invoiceAmount = regularPay + overtimePay
    else:
        invoiceAmount = totalHours * rate
    return invoiceAmount


#-------------------------------------------------------------------------------
# Author: Anoushka Das, Jaeden Patterson, Mridul Jain
# Program: ADHOC
#
# Description:
# This code reads data from a file called "Billing.txt" and processes the information for each
# employee listed in the file. For each employee, the code calculates the total billable amount
# due based on the employee's hourly rate and the total number of hours worked over a
# four-week period. It then prints a table that displays the employee's name, hourly rate, hours
# worked for each week, total hours worked, and the total billable amount due. After processing
# all employees in the file, the code calculates and prints the total billable due for all employees,
# the total billable hours worked, and the average billable hours per employee. If the file is not
# found, the code prints an error message.
#-------------------------------------------------------------------------------
import BillingModule2
```

```
def main():
    try:
        billingFile = open("Billing.txt", "r")

        print("Employee\t Rate\t Week 1\t Week 2\t Week 3\t Week 4\t Hours\t Total")

        name = billingFile.readline().rstrip("\n")

        count = 0
        totalHours = 0
        totalBillable = 0
        averageBillable = 0

        while name != "":
            count += 1
            rate = float(billingFile.readline().rstrip("\n"))
            week1 = float(billingFile.readline().rstrip("\n"))
            week2 = float(billingFile.readline().rstrip("\n"))
            week3 = float(billingFile.readline().rstrip("\n"))
            week4 = float(billingFile.readline().rstrip("\n"))
            hours = week1 + week2 + week3 + week4

            totalHours += hours

            total = BillingModule2.totalBillableDue(rate, hours)
            totalBillable += total

            print(name," \t $", format(rate, ".2f"), "\t ", format(week1, ".2f"), "\t ", format(week2, ".2f"), "\t ",\
                format(week3, ".2f"), "\t ", format(week4, ".2f"), "\t ", format(hours, ".2f"), "\t $", format(total,
",.2f"), sep="")

            name = billingFile.readline().rstrip("\n")

        billingFile.close()

        if count == 0:
            print("No employees on file")

        if count > 0:
            averageBillable = totalHours / count
            print("\nTotal Billable Due:\t $", format(totalBillable, ",.2f"), sep="")
            print("Total Billable Hours:   ", format(totalHours, ".2f"))
            print("Average Billable Hours:  ", format(averageBillable, ".2f"), "\n", sep="")

    except FileNotFoundError:
        print("Error, no file found.")
```

```
#-------------------------------------------------------------------------------
# Author: Anoushka Das, Jaeden Patterson, Mridul Jain
# Program: Menu
#
# Description:
# This Python code implements a billing system menu with three options: ending the program,
# entering billing data, and displaying an ad-hoc billing report. The program prompts the user to
# select an option from the menu and executes the corresponding task based on the option
# chosen. If the user enters an invalid option, the program prompts them to enter a valid one.
# The program uses try-except blocks to handle possible exceptions that may occur during the
# execution.
#-------------------------------------------------------------------------------

import program5
import ADHOC
import BillingModule2

BillingModule2.resetBillingFile("Billing.txt")

def main():
    again = True

    while again == True:
        print("\nBilling System Menu:\n \
          \n\t0 - end \
          \n\t1 - Enter billing data \
          \n\t2 - Display ad-hoc billing report")

        try:
            option = int(input("\nOption ==> "))

            if option == 0:
                again = False
                print("\nProgram ended successfully.")
            elif option == 1:
                program5.main()
            elif option == 2:
                ADHOC.main()
            else:
                print("Please enter an available option.\n")

        except ValueError:
            print("Select option '0' to end the menu.\n")
```

main()

Exhibit 1 - Program5

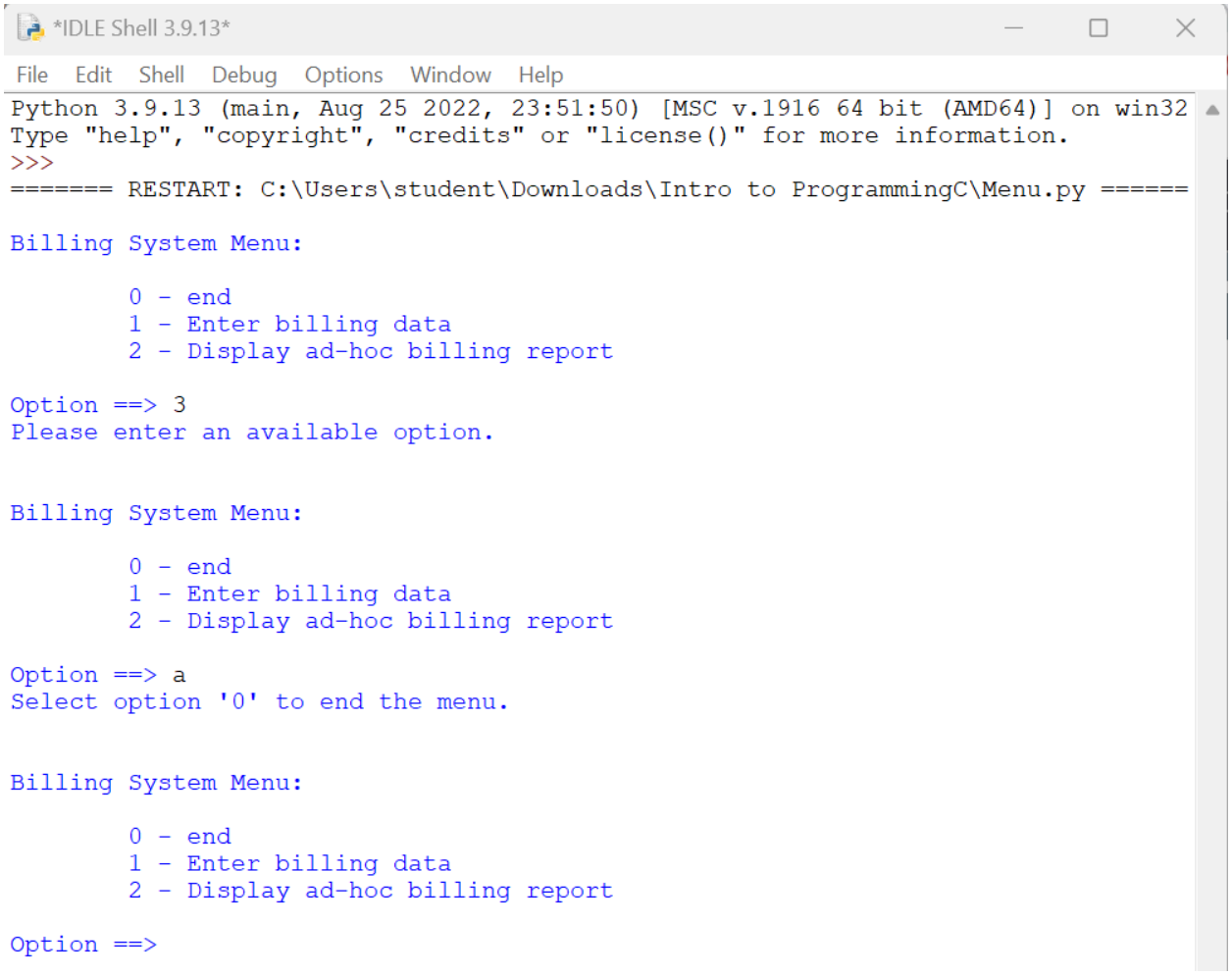

Figure 2 - Case test 1: Invalid choice from the menu

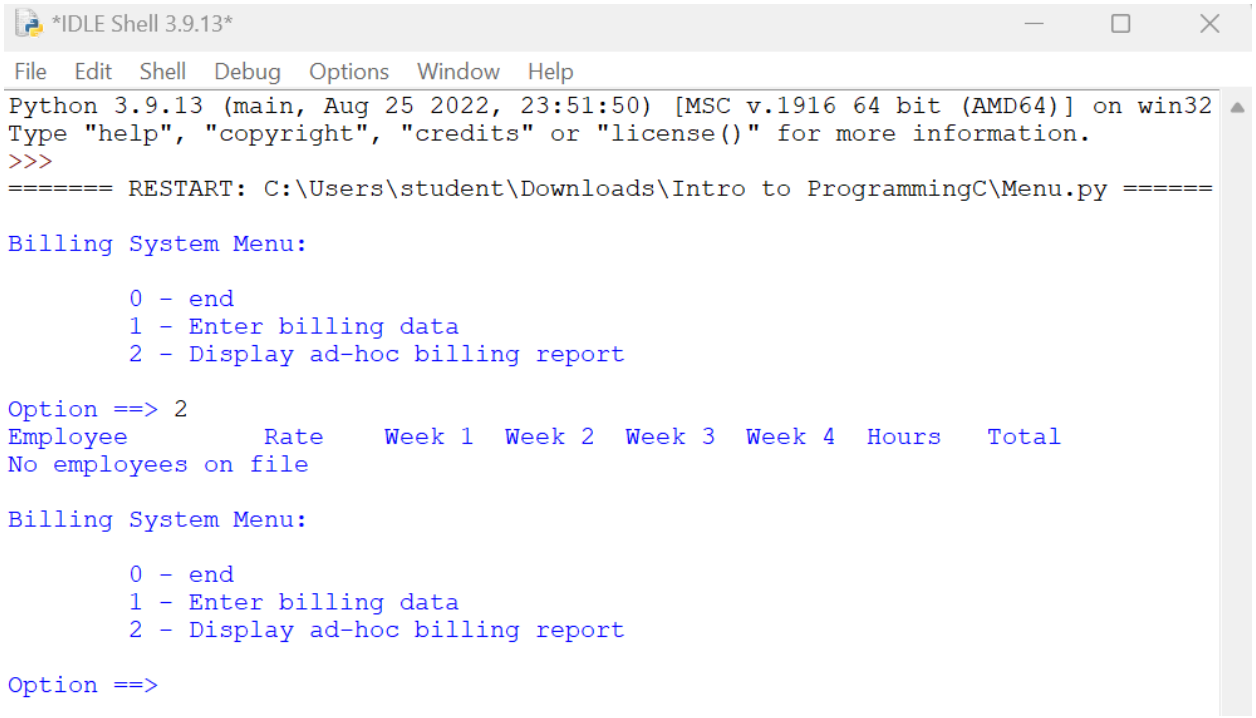

Figure 3 - Case test 2: User attempts to run the ad-hoc report without a billing.txt file

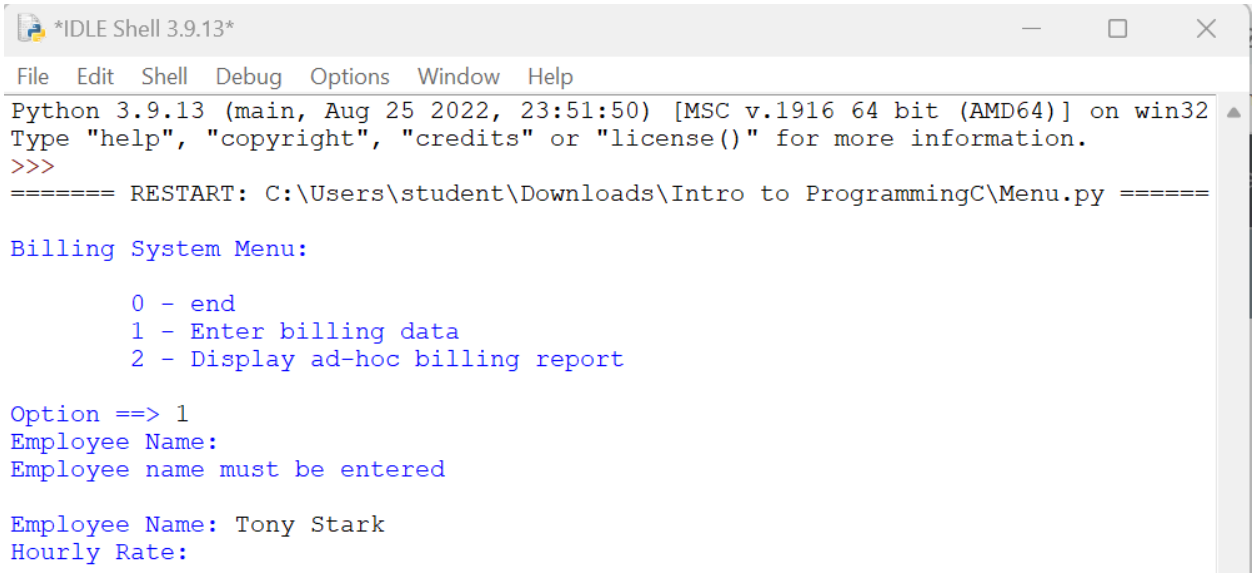

Figure 4 - Case test 3: Verify “Employee Name” validation

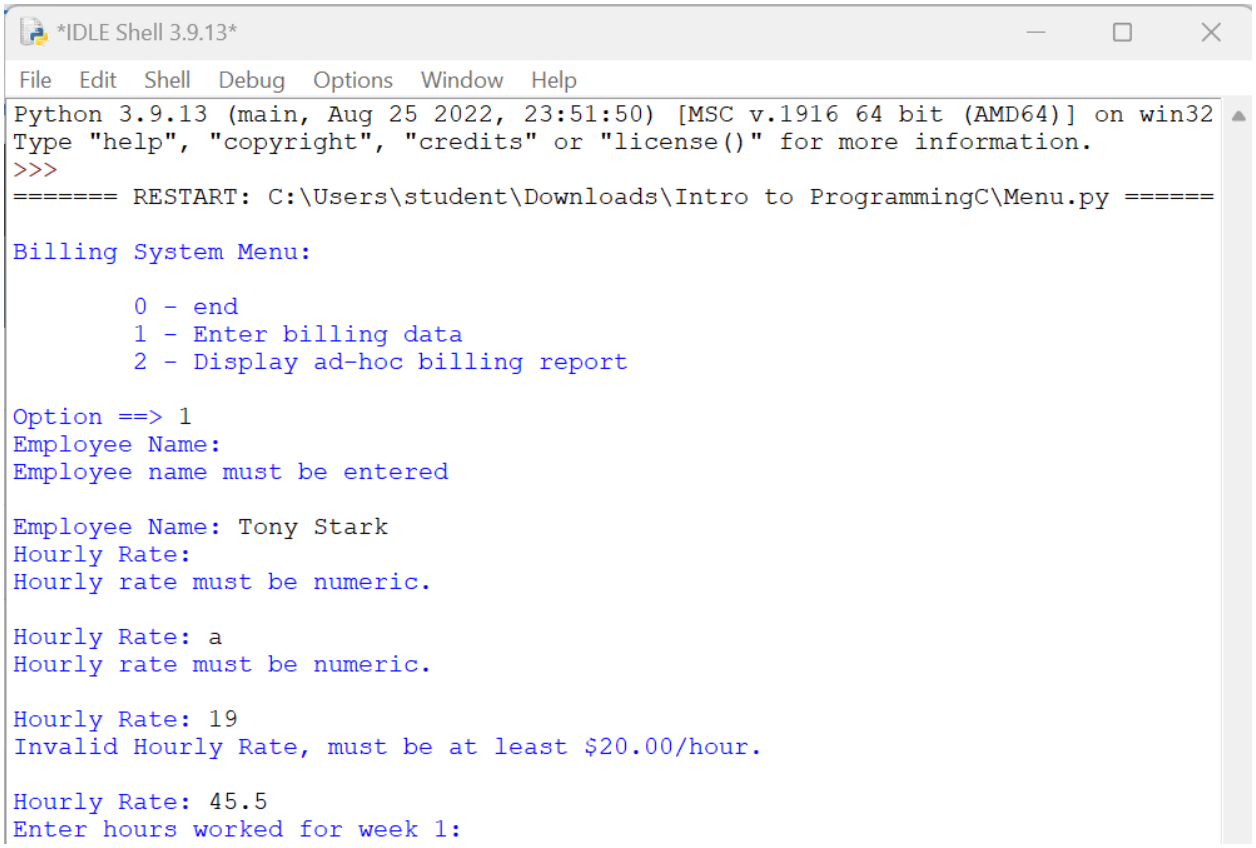

Figure 5 - Case test 4: Verify the “Hourly Rate” validation

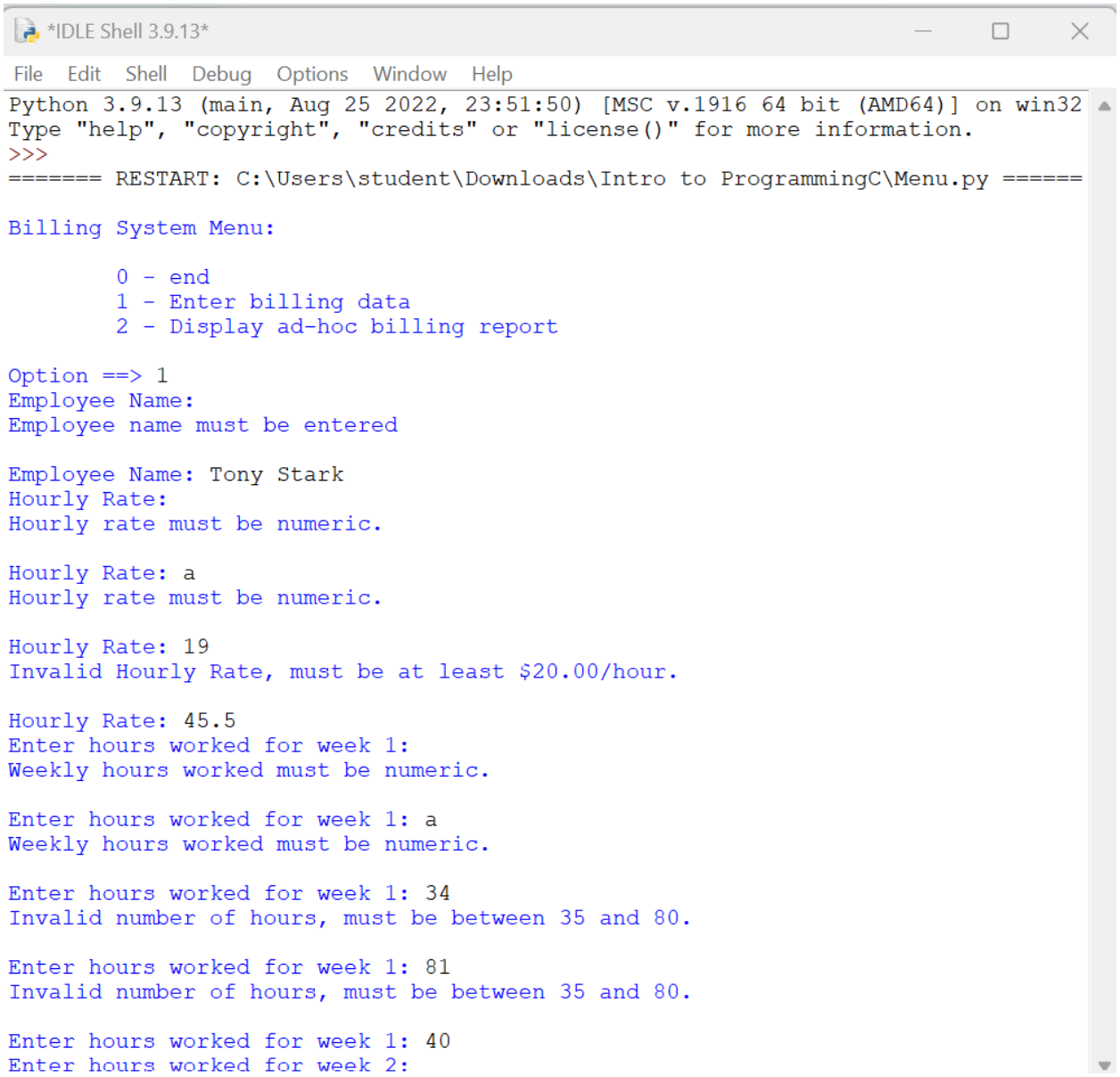

Figure 6 - Case test 5: Verify the Weekly Hours validation

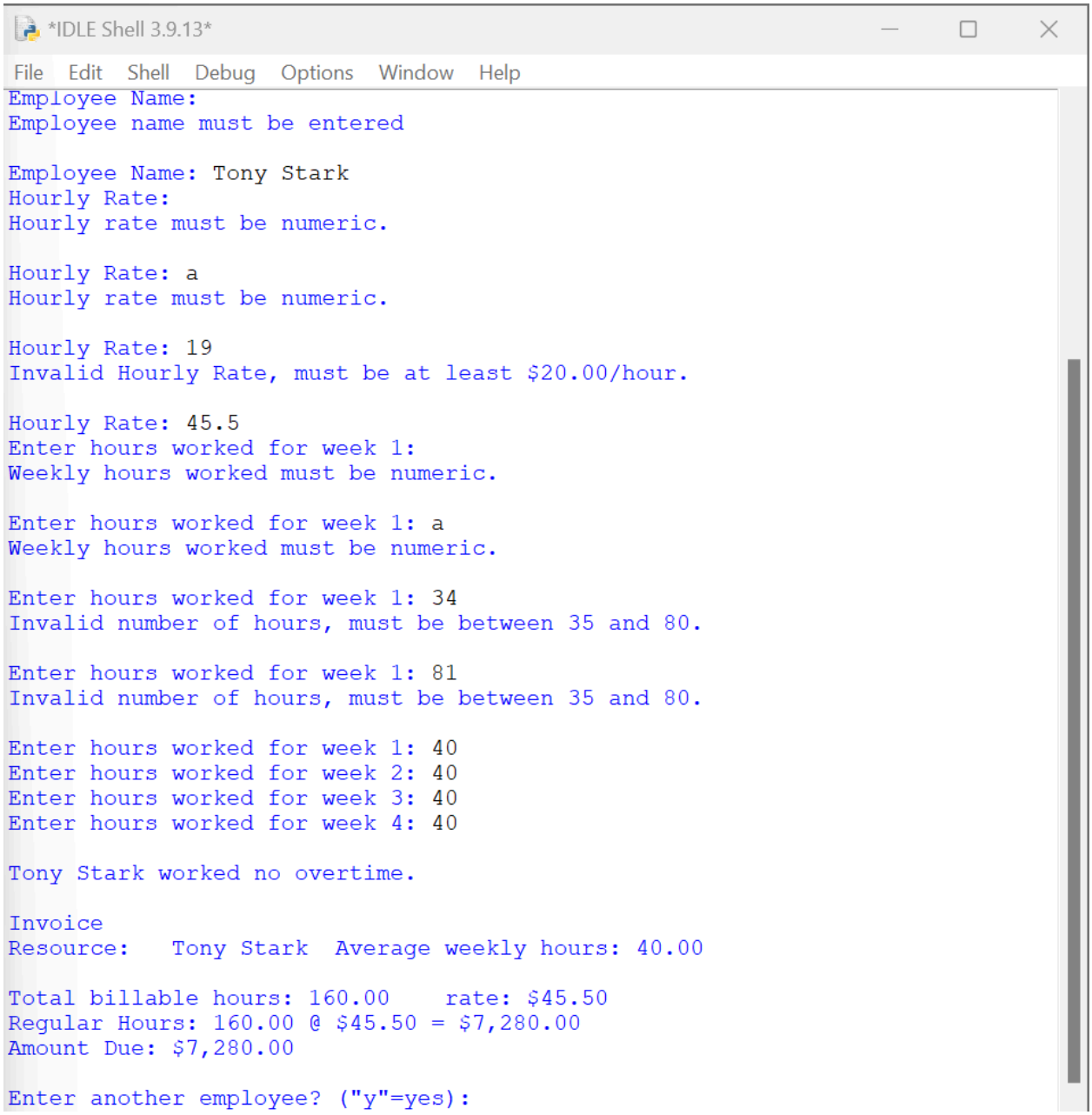

Figure 7 - Case test 6: Produce a valid invoice for a standard month (all weeks are 40 hours worked)

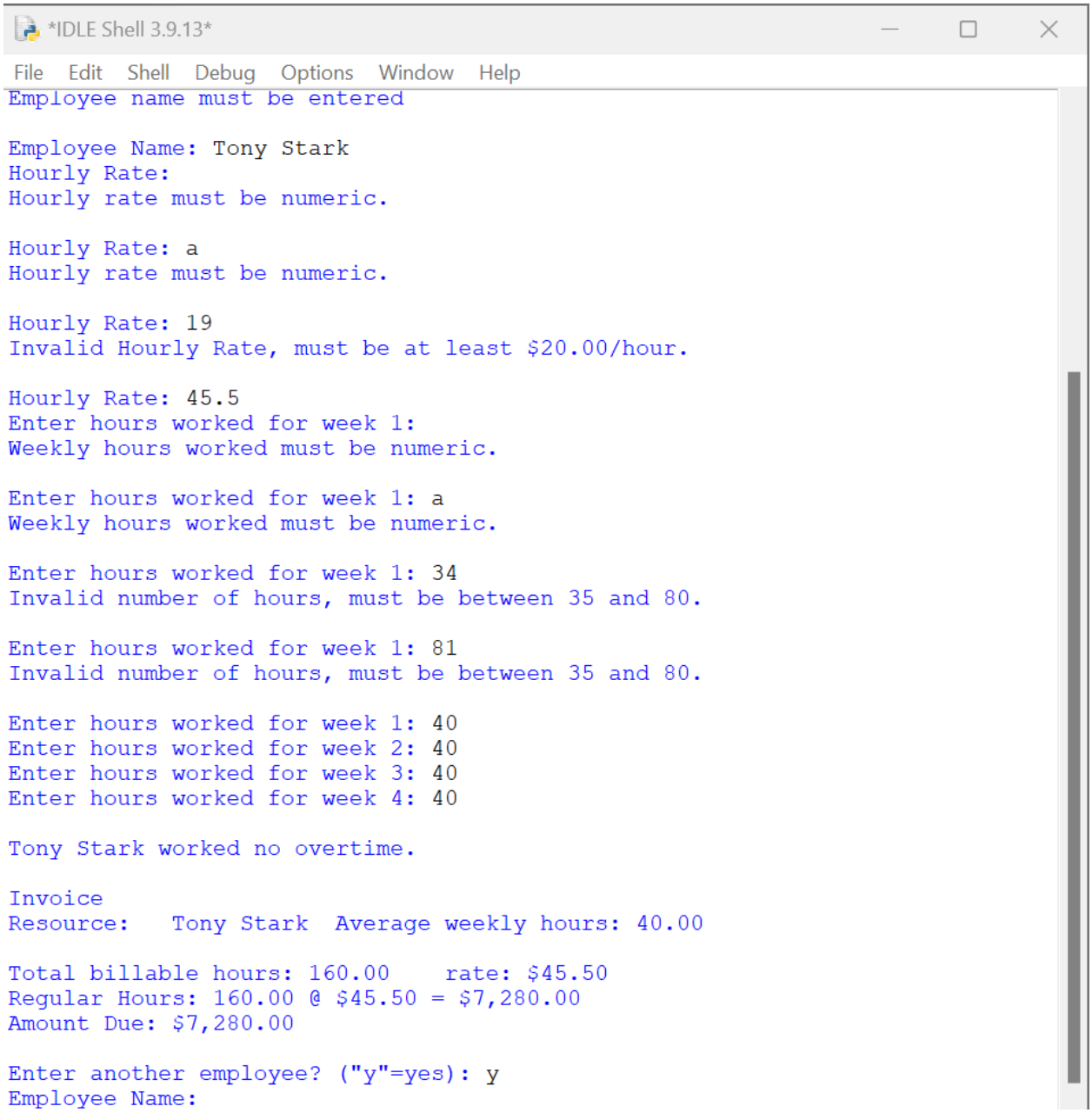

Figure 8 - Case test 7: Verify the validation for the prompt to enter another employee

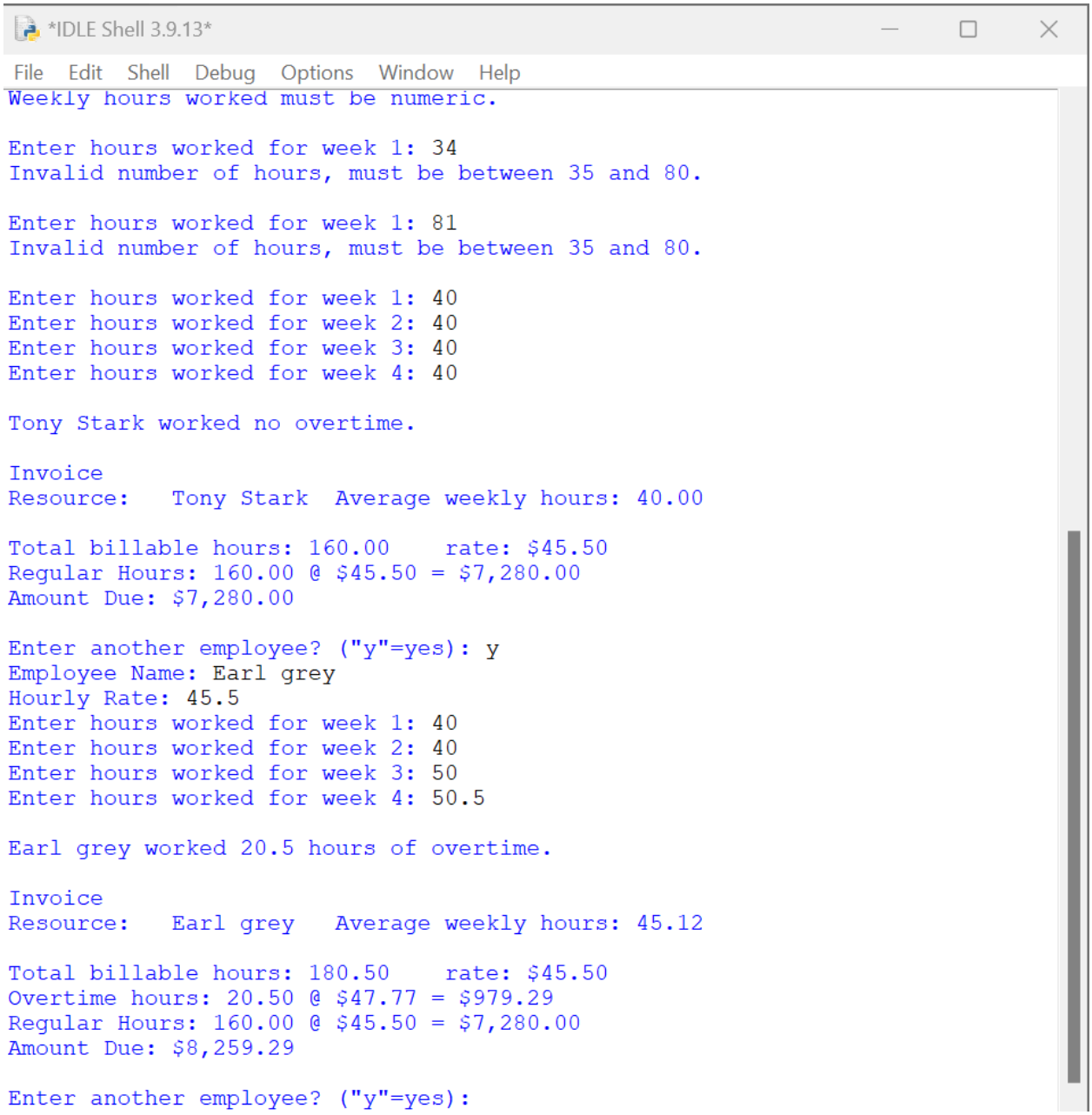

Figure 9 - Case test 8: Verify the overtime invoice is produced correctly

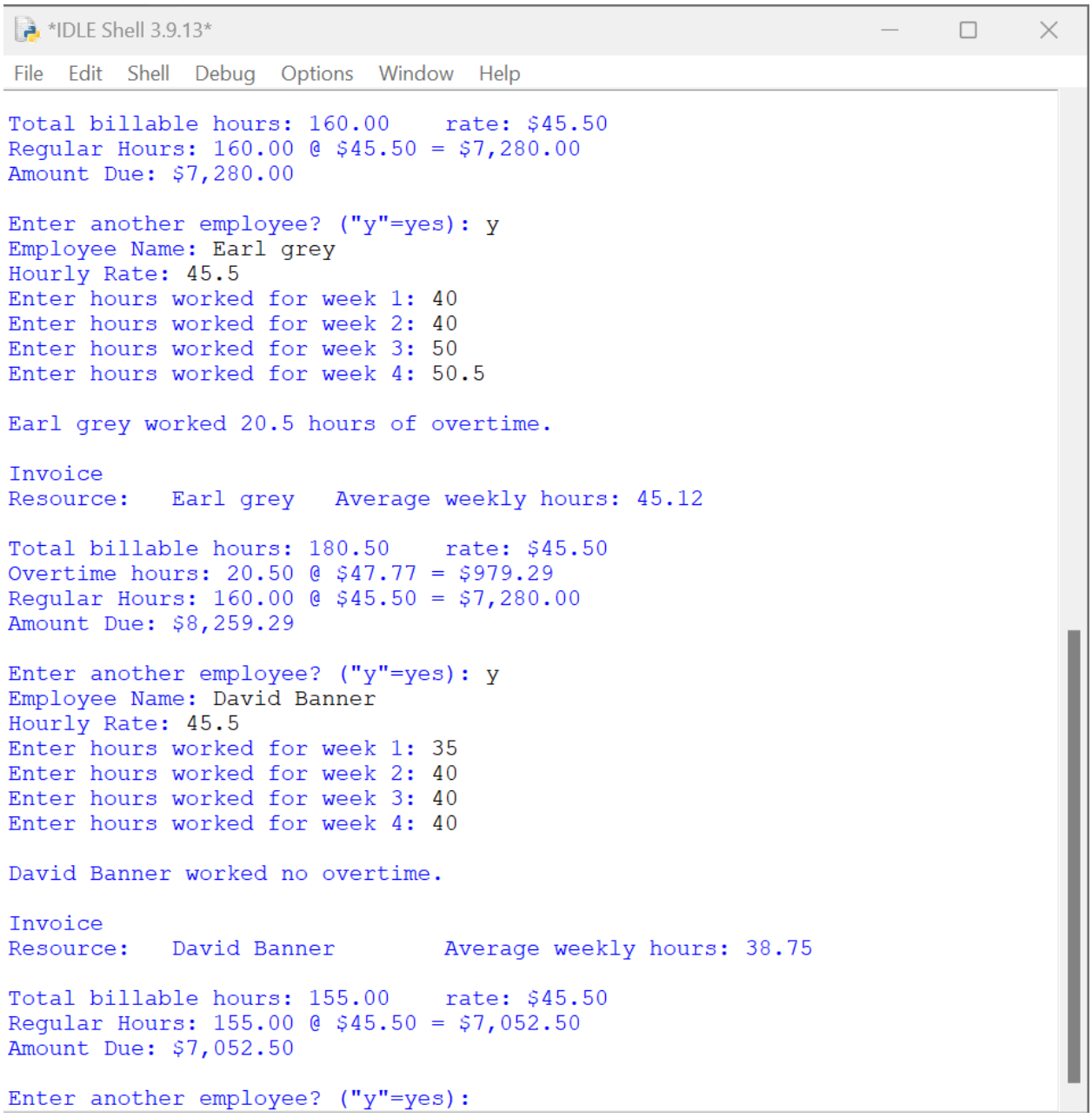

Figure 10 - Case test 9: Verify the invoice is correct for an employee with no over time working less than standard hours

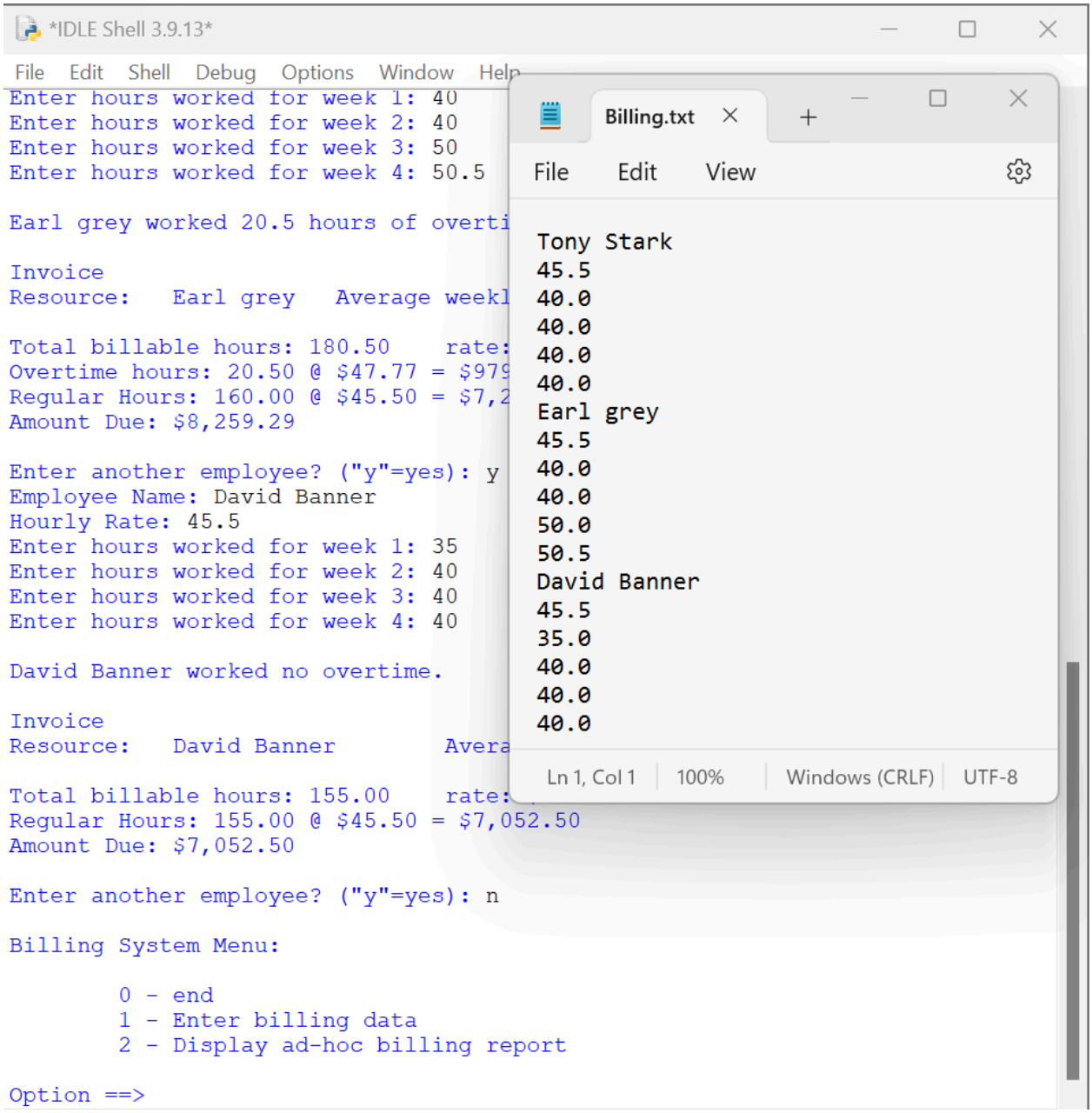

Figure 11 - Case test 10: Validate the billing.txt file is correct

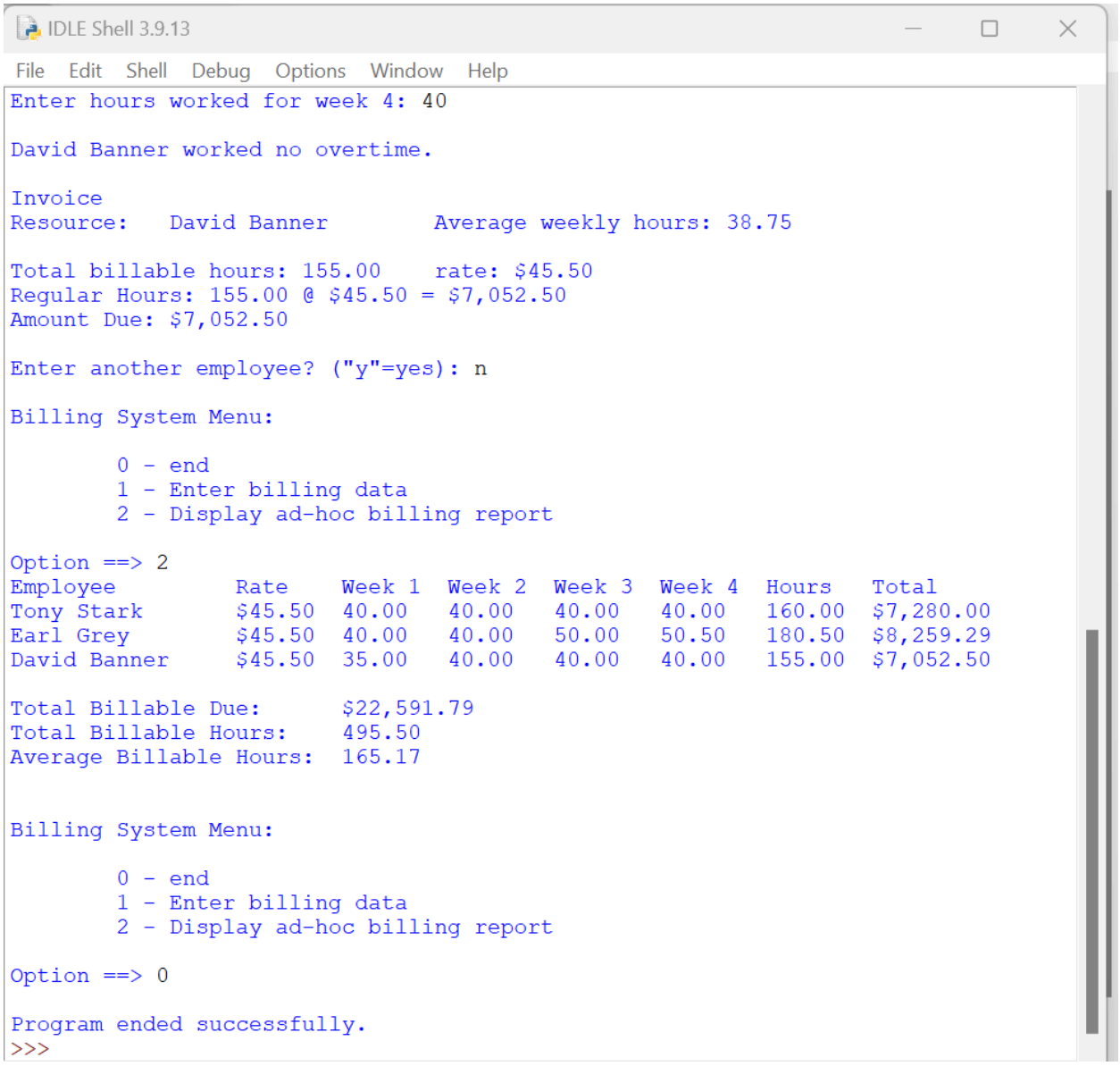

Figure 12 - Case test 11: Verify that the ad-hoc report program produces the correct report