## Learning to Use the Development Environment

Obviously you cannot program unless you understand how to create a file containing your C++ language source code, how to compile and link the source code to produce a program, how to execute (run) the program, test its behavior, fix errors, etc. For the first programming project, you will be given C++ source for a program that meets the specification given below. Your task is to type in the given source code, without making any unspecified modifications, and verify that it does indeed perform as specified. Along the way, you'll be exposed to quite a bit of C++ language that we haven't even begun to cover, so don't get paranoid. Try to understand the given source code (I've included lots of comments to help), but realize that this program uses elements of the C++ language from the first six sections of the class notes.

## What to turn in and how:

This project will be submitted to the New Automated Grader (NAG). Instructions for submitting to NAG are available from the NAG homepage; see the link from the CS 1044 web page. Be sure to follow those instructions carefully. In particular, do not alter the labels used in the sample output file below, and do not produce additional lines of output for aesthetic reasons.

## The Program Specification: Decision Making, Repeated Actions and File I/O: the Macro$oft Corporation Payroll

You are to write a very simple sort of payroll program, which will read data from a file, perform some arithmetic calculations, and write results in a nice tabular form to a file. The forms of the input and output files are described below, with examples. Be sure to pay careful attention to both.

## Input file description and sample:

Your program **must** read its input from a file named `workers.dat` — use of another input file name will result in massive deductions. Each line of the input file will contain four values, separated by whitespace:

- Employee ID number (integer between 1000 and 9999)
- Employee's age (integer between 18 and 99)
- Employee's gross salary (real number between 0.0 and 9999.99)
- Insurance policy type (character – either B or D)

You may assume that all input values will be logically correct (no negatives, for instance), and that all values will be in the specified ranges. For instance:

```
7378      28   1331.00    B
2211      60   4532.00    D
4129      47   2305.00    D
6039      59   3981.00    D
4559      70    565.00    B
9912      29   2728.00    B
```

Note that you must not make **any** assumptions about the number of lines of data in the input file. Your program must be written so that it will detect when it's out of input and terminate correctly (a technique for this will be presented in class).

## What to Calculate:

You will write a program which computes the net monthly salary of each employee from his/her gross monthly salary. The deductions made on monthly gross salary are for insurance and taxes. The insurance fee paid by each employee is determined from the first table below. Each employee must also have amounts withheld for income and social security taxes according to his/her gross monthly income, at rates given in the second table:

| Age | Monthly Insurance Cost | |
|---|---|---|
| | Plan B | Plan D |
| **up to 35** | $110 | $140 |
| **36 to 65** | $160 | $220 |
| **over 65** | $250 | $400 |

| Gross Monthly Income | Income Tax Rate | Social Security Tax Rate |
|---|---|---|
| up to $999.99 | 15% | 6.83% |
| $1000.00 to $2999.99 | 28% | 6.83% |
| $3000.00 and up | 33% | 6.83% |

You must also calculate the average gross salary, average insurance fee, average income tax withheld, average social security tax withheld, and average net salary for all the employees for whom you are given data.

## Output description and sample:

Your program must write output data to a file named `payroll.dat` — use of any other output file name will result in a massive deduction of points.

The sample output file shown below corresponds to the input data given above:

```
Bill McQuain
Macro$oft Corporation Payroll

 IdNum   Gross Pay    F.I.T.       SSI      Ins.     Net Pay
============================================================
  7378     1331.00    372.68     90.91    110.00      757.41
  2211     4532.00   1495.56    309.54    220.00     2506.90
  4129     2305.00    645.40    157.43    220.00     1282.17
  6039     3981.00   1313.73    271.90    220.00     2175.37
  4559      565.00     84.75     38.59    250.00      191.66
  9912     2728.00    763.84    186.32    110.00     1667.84
============================================================
  Avg:     2573.67    779.33    175.78    188.33     1430.23
```

You are not required to use this exact horizontal spacing, but your output must satisfy the following requirements:

- All dollar amounts must be printed to show two places after the decimal.
- You must use the specified header, and column labels, and include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column.
- You must show the required averages below the corresponding columns.
- You must use the same ordering of the columns as shown here.

## Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. A copy of *Elements of Programming Style* is included with the course notes — if you don't have a copy I strongly suggest you read the on-line edition (available from the course web page). Some specifics:

- You must include header comments specifying the compiler and operating system used and the date completed.
- You must include a comment explaining the purpose of every variable you use in your program.
- You must use meaningful, suggestive (of function or purpose!) variable names.
- Use named constants instead of variables where appropriate.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation to make control structures like loops and if-else statements more readable.

## Hints:

This program requires that you know how to manage file-oriented input/output operations --- the slides and the text provide good examples and guidelines. You'll have to use **manipulators** to manage the formatting of your code. Read the discussion in the course notes carefully, there are important clues there.

You'll need to use if-then control structures extensively --- be sure to read Chapter 5 of the course notes. You'll also need to use some sort of loop control structure to manage reading and processing of input. That's covered in Chapter 6 of the course notes.

## The Source Code:

```
//  Payroll98.cpp:  A Simple Payroll Program
//
//  You should substitute the correct information in the following
//  lines:
//
//  Programmer:   Bill McQuain
//  ID Number:    999-99-9999
//  Compiler:     Visual C++ version 5.0
//  Platform:     Pentium 200 / Windows NT
//  Date:         August 13, 1998
//
//  Purpose of the program:
//
//  This program reads information about an arbitrary number of
//  employees from an input file named "workers.dat":
//
//    - ID number
//    - age
//    - gross monthly salary
//    - insurance plan type
//
//  and computes the correct deductions for:
//
//    - federal income tax
//    - social security tax
//    - insurance fee
//    - net monthly salary.
//
//  It then prints out a labeled table of results, showing for
//  each employee the ID number, gross salary, insurance fee,
//  income and social security tax deductions and net salary.
//  In addition, the averages of each category are computed and
//  printed.
//
//  The is the #include section.  These statments tell the compiler
//  to read and use variables and functions declared in the specified
//  files.
#include <fstream.h>
#include <iomanip.h>

//  Every C++ program must have a function named main.  This is the
//  beginning of the definition of the main function for this
//  simple program.  In this case the whole program consists of just
//  the main function --- usually programs involve more than one
//  function.
//
int main() {

//  This section is where named constants are declared and given
//  their values.  We could just use the numbers themselves in the
//  code that follows, but the names make the code more readable;
//  the name SSRate carries meaning where the number 0.0683 doesn't.
//

  const double HiTax  = 0.33;           // tax rates
  const double MidTax = 0.28;
  const double LoTax  = 0.15;
  const double SSRate = 0.0683;
```

```
  const double HiPlanB  = 250.00;       // insurance fees
  const double MidPlanB = 160.00;
  const double LoPlanB  = 110.00;
  const double HiPlanD  = 400.00;
  const double MidPlanD = 220.00;
  const double LoPlanD  = 140.00;

  const double HiIncome  = 3000.00;     // income levels
  const double MidIncome = 1000.00;

  const char  PlanB = 'B';              // insurance plan types
  const char  PlanD = 'D';


//  This section is where the variables used in main() are declared.
//  Each variable has a type (integer, double, etc) but no particular
//  value at this point.  The variable names like the constant names
//  used above are supposed to be meaningful, and each declared
//  variable is also given a short comment explaining what it will
//  be used for.
//
  ifstream inPay;          // input file stream
  ofstream outPay;         // output file stream

  int   IdNum,             // employee ID number,
        Age,               //          age.
        NumEmployees;      // number of employees.
  double GrossPay,         // employee gross pay.
         NetPay,           //          net pay,
         SSI,              //          SS tax,
         FIT,              //          income tax,
         InsFee,           //          insurance fee..
         TotalGross,       // total of all gross pay,
         TotalNet,         //              net pay,
         TotalIns,         //              insurance fees,
         TotalSSI,         //              SS tax,
         TotalFIT;         //              income tax.
  char   InsPlan;          // employee insurance plan

//  This is the initialization section of main().  We must make sure that
//  every variable is given a value before it is used in a calculation
//  or printed.  Some variables get their values by being read from the
//  input file, so they don't need to be initialized here.

  NumEmployees =   0;      // Initialize counters and running totals
  TotalGross   = 0.0;      // to zero.
  TotalNet     = 0.0;
  TotalIns     = 0.0;
  TotalSSI     = 0.0;
  TotalFIT     = 0.0;

//  The following four statements prepare the input and output files
//  for use.  Don't worry too much about what's going on here just yet,
//  we'll cover the details shortly.
//
  inPay.open("workers.dat");       // open input and output files
  outPay.open("payroll.dat");
  outPay.setf(ios::fixed, ios::floatfield);
  outPay.setf(ios::showpoint);
```

```
//  The following statments print the header info to output file.  It's
//  rather cryptic at this point, but you'll see output is really very
//  simple in C++.

  outPay << "Bill McQuain" << endl;
  outPay << "Macro$oft Corporation Payroll" << endl << endl;
  outPay << " IdNum   Gross Pay    F.I.T.       SSI      Ins.     Net Pay";
  outPay << endl;
  outPay << "============================================================";
  outPay << endl;

//  This statement tries to read first line of data from input file.  If
//  something goes wrong, the variable inPay will be false (small lie) and
//  that will prevent us from going into the loop that follows and generating
//  nonsense.

  inPay >> IdNum >> Age >> GrossPay >> InsPlan;

//  What comes next is a while loop.  The body of the loop gets
//  executed over and over again until the expression that follows
//  the while (inPay in this case) becomes false.  Therefore, we'd
//  better be sure that inPay will eventually become false,
//  otherwise the program will continue to execute the body of this
//  loop forever.

  while (inPay) {

        //  The first part of the loop body (down to the comment with
        //  all the hyphens) processes the data that was just read in.

        NumEmployees++;                   // Count employees.
        TotalGross += GrossPay;           // Update total gross pay.
        inPay.ignore(200, '\n');          // Skip to next line of input.

        //  The switch statement tries to match the value of the selection
        //  variable (InsPlan this time) with the given cases.  When a
        //  match is found, the statements that accompany that case are
        //  executed.  If no match is found, the default case is carried out.

     switch (InsPlan) {                // Calculate insurance fees:
     case 'B':{if (Age <= 35)          //
                 InsFee = LoPlanB;     //    for Plan B employees
               else if (Age <= 65)
                 InsFee = MidPlanB;
               else
                 InsFee = HiPlanB;
               break;
              }
     case 'D':{if (Age <= 35)          //    for Plan D employees
                 InsFee = LoPlanD;
               else if (Age <= 65)
                 InsFee = MidPlanD;
               else
                 InsFee = HiPlanD;
               break;
              }
     default :{outPay << "Employee " << setw(4) << IdNum;
               outPay << " has invalid insurance plan." << endl;
               InsFee = 0.0f;
              }
     }
```

```
      TotalIns += InsFee;            // Update total insurance fees.

      if (GrossPay >= HiIncome) {       // Determine FIT amount.
         FIT = HiTax * GrossPay;
      }
      else if (GrossPay >= MidIncome) {
         FIT = MidTax * GrossPay;
      }
      else {
         FIT = LoTax * GrossPay;
      }
      TotalFIT += FIT;               // Update total income taxes.

      SSI = GrossPay*SSRate;         // Calculate SS tax.
      TotalSSI += SSI;               // Update total SS taxes.

      NetPay = GrossPay - InsFee -
                                FIT - SSI;  // Calculate net pay.
      TotalNet += NetPay;                   // Update total net pay.

      // --------------------------------- End of Processing Section

      //  This section of the loop body prints the results calculated
      //  above into the output file.

      outPay << setw( 6) << IdNum;
      outPay << setw(12) << setprecision(2) << GrossPay;
      outPay << setw(10) << setprecision(2) << FIT;
      outPay << setw(10) << setprecision(2) << SSI;
      outPay << setw(10) << setprecision(2) << InsFee;
      outPay << setw(12) << setprecision(2) << NetPay;
      outPay << endl;

      //  This statement tries to read the next line of input.  This
      //  is the last statement in the loop body, so the next thing
      //  that will happen is that the loop control expression (inPay
      //  in this case) is tested.

      inPay >> IdNum >> Age >> GrossPay >> InsPlan;

   }  // End of while (inPay)

//  Once we've exited the loop, we're done reading and processing data for
//  new employees.  It's time to sum things up.  The following statements
//  print out the averages in a nice form.  First print a line to mark
//  the end of the table body:

   outPay << "============================================================";
   outPay << endl;
```

```
//  Now, calculate and print averages, if appropriate.  We've got to be
//  careful not to calculate averages if there weren't any employees, so
//  this uses an if statement to be sure that's not the case before any
//  dividing is done.

  if (NumEmployees != 0) {
     outPay << "  Avg:";
     outPay << setw(12) << setprecision(2) << TotalGross/NumEmployees;
     outPay << setw(10) << setprecision(2) << TotalFIT/NumEmployees;
     outPay << setw(10) << setprecision(2) << TotalSSI/NumEmployees;
     outPay << setw(10) << setprecision(2) << TotalIns/NumEmployees;
     outPay << setw(12) << setprecision(2) << TotalNet/NumEmployees;
     outPay << endl;
  }

//  These two statements just close the input and output files; this
//  tells the operating system that we're done with them and (hopefully)
//  that the files are properly saved.

  inPay.close();
  outPay.close();

//  The return statement sends a value back to whoever called this
//  function.  If you're using Visual C++, you may see this value
//  displayed in the window at the bottom of the screen when you run
//  the program.

  return NumEmployees;
}
```