## Payroll Application Description

### Overview:

Payroll application is a simple payroll program that allows a company to create a payroll database for their staff. It is also pre-loaded with sample company data for a company called “default”.

The application is a self-contained executable; however it will use and generate output files in its program directory. Including a database file for the company, a csv file as an output option, and a html file as an output option. It also uses temp files for file operations.

For developers wanting to improve on the application the source code is included in the \assignment2-new folder and includes:   
- main.cpp: the main program containing program flow  
- payroll.cpp: all program functions (excluding main)  
- payroll.h: function definitions and program constants for the application  
- Four class implementation files named: company.cpp, manager.cpp, staffmember.cpp and casual.cpp  
-Four class header files named: company.h, manager.h, staffmember.h and casual.h

### Program Description:

The program launches into an option to load an existing database. If the user enters no, it is assumed they would like to work with the default data in the application, and the default.db database will be used for the application. If the user enters yes, they will be prompted to enter the company name they want to work with. All registered company names are stored in “companies.db”. If the company entered does not exist, the user is prompted to decide if they want to create a database for that company. If yes, the database is created, and companies.db is updated. If no, the user is free to work with the default.db.

The program is split into five main code paths:  
- Add New Staff Member  
-Display or Edit Staff Member  
-Remove Staff Member  
-Report on Staff Members  
-Exit Program

Add New Staff Member:

Uses the function “getnextid” to obtain the next usable staff id. Staff ids are unique, and are set via program logic, not by a user. Once the id is obtained “createStaffMember” is called. It uses the four classes company, manager, staffmember and casual to build a new staffmember object which is then commited to the database at the end. The staffmember object is the parent for the child classes called casual and manager, and most functionality of the manager and casual classes is inherited.

Display or Edit Staff Member:

The “displaystaffMember” function is called to give the user a chance to output details of a previously created staffmember, this is outputted to the screen. Then the user is given an option to edit this staff member – if they decide to edit the “editstaffmember” function is called. It takes the id of the staffmember as a parameter and uses this to slice the edited staff member back into the database in the right location.

Remove Staff Member:

Single function called “removestaffmember” is called. The staff member is first displayed to the user so they can ensure they have the right one, then they are given the opportunity to confirm they want to delete. If they delete, the staff member is taken out of the database. The user must confirm database changes at application close, so it is still possible at this point to recover from a mistake.

Report on Staff Members:

The “reportAllStaff” function displays a report of all users wages for a week, as well as a total. It then presents users with an option to export to csv format for use in other applications, or to output to a html web page for display – both options are implemented in their own functions (“exporttocsv” and “exporttohtml”).

Exit Program:

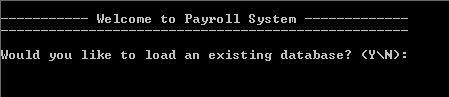
The exit program option does not just close the application. At this point the user decides if changes to their database should be committed. If the user chooses yes, the database is left alone. If the user chooses no, the previous copy of the database is rolled back in to the live copy and data is as it was when the program launched.

### Developers Notes:

Lots of small functions outside the main code have been included for things such as converting from integer types, to strings and vice versa. Also things such as checking for valid input have been made into small function that accept the input as a parameter and return if they are valid. All are designed to save code repetition. Other areas of code could be improved and modularised, but time constraints meant this could not be achieved. In v2 of this application we could cut down especially the file IO operations by using functions with “override” functionality where the number of parameters decided the code path in the function.

### Sample Program Output

Program Launch:



Main Screen:

