**C/C++ SYNTAX, LOGIC AND RUNTIME ERRORS LAB REPORT**

**1) Enter your name, student ID, platform (Mac or PC) and date**

Name and StudentID: Samuel Indurkar, 0888068

Class: CIS054 C/C++ Programming

Platform (Mac or PC): Mac gcc and eclipse

Date: 6/20/2017

**OBJECTIVES:**

Create syntax, logic and runtime errors. Observe, document and analyze the program execution.

**PROJECT DESCRIPTION:**

Modify the Payroll program to create a collection of errors as shown in the tables below. A **syntax** error is a violation of the rules of the language. Sometimes a syntax error is not serious enough to prevent the program from being compiled into machine code, so a warning is issued instead of an error. As a matter of good programming practice, you should clear up the warnings even though the program compiles and runs.

**LAB REPORT:**

**2) Fill in the second column of the table.**

After you have created each of the following errors or warnings, describe the result of the error. **Restore the program to a running state before you proceed and create the next error or warning.** It may even be a good idea to make a backup copy of your program code before you even start messing around with it to intentionally create errors.

|  |  |
| --- | --- |
| **SYNTAX ERRORS** | **DESCRIBE WHAT HAPPENED** |
| Put an extra space between the **<** and the **iostream** file name | I don’t have iostream because I didn't write cpp. I wrote c program, so I put an extra space between the < and stdio.h. “Payrole\_Indurkar.c:7:10: fatal error: ' stdio.h ' file not found  #include < stdio.h >  **^**  1 error generated.  “ |
| Leave off one of the < or > characters in the **#include** statement | **Payrole\_Indurkar.c:7:18: error: expected "FILENAME" or <FILENAME>**  #include <stdio.h  **^**  **It requires a close bracket or ‘ “ ‘** |
| Leave off the **int** from  **int main** | Only a warning is generated the program may still function.  **Payrole\_Indurkar.c:9:2: warning: type specifier missing, defaults to 'int'**  **[-Wimplicit-int]**  main(int argc, char\* argv[])  **^**  1 warning generated. |
| Leave off, or misspell **main** | doesn’t know where to start, couldn’t find main.    Undefined symbols for architecture x86\_64:  "\_main", referenced from:  implicit entry/start for main executable  ld: symbol(s) not found for architecture x86\_64  clang: error: linker command failed with exit code 1 (use -v to see invocation) |
| Leave off one of the **(** or ) then omit both the  **( )** | it expects a matching bracket  **Payrole\_Indurkar.c:28:2: error: expected ')'**  {  **^**  **Payrole\_Indurkar.c:27:5: note:** to match this '('  if (hours <= 40  **^**  1 error generated.  **Payrole\_Indurkar.c:27:5: error: expected '(' after 'if'**  if hours <= 40  **^**  1 error generated. |
| Leave off the closing curly brace **{** | it expects a matching bracket  **Payrole\_Indurkar.c:32:2: error: expected identifier or '('**  else  **^**  **Payrole\_Indurkar.c:37:2: warning: type specifier missing, defaults to 'int'**  **[-Wimplicit-int]**  regpay = reghours \* payrate;  **^**  **Payrole\_Indurkar.c:37:11: error: use of undeclared identifier 'reghours'**  regpay = reghours \* payrate;  **^**  **Payrole\_Indurkar.c:37:22: error: use of undeclared identifier 'payrate'**  regpay = reghours \* payrate;  **^**  **Payrole\_Indurkar.c:38:2: warning: type specifier missing, defaults to 'int'**  **[-Wimplicit-int]**  overtimepay = overtimehours \* payrate \* 1.5;  **^**  **Payrole\_Indurkar.c:38:16: error: use of undeclared identifier**  **'overtimehours'; did you mean 'overtimepay'?**  overtimepay = overtimehours \* payrate \* 1.5;  **^~~~~~~~~~~~~**  overtimepay  **Payrole\_Indurkar.c:38:2: note:** 'overtimepay' declared here  overtimepay = overtimehours \* payrate \* 1.5;  **^**  **Payrole\_Indurkar.c:38:32: error: use of undeclared identifier 'payrate'**  overtimepay = overtimehours \* payrate \* 1.5;  **^**  **Payrole\_Indurkar.c:39:2: warning: type specifier missing, defaults to 'int'**  **[-Wimplicit-int]**  paycheck = regpay + overtimepay;  **^**  **Payrole\_Indurkar.c:39:20: error: initializer element is not a compile-time**  **constant**  paycheck = regpay + overtimepay;  **~~~~~~~^~~~~~~~~~~~~**  **Payrole\_Indurkar.c:42:10: error: expected parameter**  **ma declarator**  printf ("your pay is $%0.2f\n", paycheck);  **^**  **Payrole\_Indurkar.c:42:10: error: expected ')'**  **Payrole\_Indurkar.c:42:9: note:** to match this '('  printf ("your pay is $%0.2f\n", paycheck);  **^**  **Payrole\_Indurkar.c:42:2: warning: type specifier missing, defaults to 'int'**  **[-Wimplicit-int]**  printf ("your pay is $%0.2f\n", paycheck);  **^**  **Payrole\_Indurkar.c:42:2: error: conflicting types for 'printf'**  **/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.11.sdk/usr/include/stdio.h:259:6: note:**  previous declaration is here  int printf(const char \* \_\_restrict, ...) \_\_printflike(1, 2);  **^**  **Payrole\_Indurkar.c:44:2: error: expected identifier or '('**  return 0;  **^**  **Payrole\_Indurkar.c:45:1: error: extraneous closing brace ('}')**  }  **^** |

NOTE: This lab assignment contains more than one page.

A **logic** error is an error in which the program compiles and runs with a computation error and does not produce the correct or desired result. The program is doing what it was told to do, not what the programmer really wanted it to do. It is for this reason that we need to use a collection of test data values with expected results to verify that the program is functioning correctly.

The Paycheck lab contained a set of test data values and expected results.

|  |  |
| --- | --- |
| **TEST DATA VALUES** | **EXPECTED RESULT** |
| Hours = 39, PayRate = 20 | $780.00 |
| Hours = 40, PayRate = 20 | $800.00 |
| Hours = 41, PayRate = 20 | $830.00 |

For each entry in the table below (number 1, 2 and 3),

A - **Identify** whether 39 hours, 40 hours or 41 hours produced the correct or incorrect result

B - If the result is incorrect, **describe why**

|  |  |
| --- | --- |
| **LOGIC ERRORS** | **DESCRIBE WHAT HAPPENED** |
| Change the < to a > on the line:  if (Hours <= 40)  to:  if (Hours >= 40) | 1. Hours = 39, PayRate = 20   Incorrect result.  pay is $770.00. It is $10 less. Because instead of using base rate, it is using overtime rate.  2) Hours = 40, PayRate = 20 Correct result. As expected. because of =  3) Hours = 41, PayRate = 20  Incorrect result.  pay is $820.00. It is $10 less. Because instead of using overtime rate, it is using standard base rate. |
| Change the - to a + on the line:  OvertimeHours = Hours-40;  to:  OvertimeHours = Hours+40; | 1. Hours = 39, PayRate = 20 Correct result, because the logic error is only in the else case for overtime calculation   2) Hours = 40, PayRate = 20 Correct result, because the logic error is only in the else case for overtime calculation  3) Hours = 41, PayRate = 20 Incorrect result, because the logic error is in the else case for overtime calculation. It adds 40 hours extra. |
| Change the + to a \* on the line:  Paycheck = RegPay + OvertimePay;  to:  Paycheck = RegPay \* OvertimePay; | 1) Hours = 39, PayRate = 20 Incorrect result, because whenever overtime pay is 0, and we multiply it, the result is 0 2) Hours = 40, PayRate = 20 Incorrect result, because whenever overtime pay is 0, and we multiply it, the result is 0 3) Hours = 41, PayRate = 20 Incorrect result, because instead of adding the overtime pay, we are multiplying it. That is incorrect |

A **runtime** error is an error in which the program compiles and runs but crashes if bad data is input or the processor is not able to process the data (example: divide by zero)

In the table below, describe what happens for each runtime error when letters are entered for the PayRate and for the Hours

|  |  |
| --- | --- |
| **RUNTIME ERROR** | **DESCRIBE WHAT HAPPENED** |
| Restore the program to read:  if (Hours <= 40) | 1) Hours = 40, PayRate = **twenty**  2) Hours = **forty**, PayRate = 20  In both cases, the output was 0 because in both cases, it was expecting numeric input, but we entered it in words (twenty and forty) . |

**DISCUSSION:**

**3) Complete the DISCUSSION section. It does not need to be long, but it needs to be complete.**3a) Even though there was a logic error in some of the examples, why were incorrect results displayed sometimes and other times correct results displayed?

Sometimes the logic errors didn’t affect some cases. For example when we changed the <=40 to >=40, the 40 hours scenario didn’t change due to the equals sign. However the logic error did affect 39 and 41

3b) Is it good enough to just give instructions to the user that only numeric data should be input for Hours and PayRate, or should the program be designed to process unexpected inputs? Why or why not?

Yes the program should be built to accept unexpected inputs such as people spelling out the number instead of writing it (2 vs. two).