Practical 5

COS132

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Official Deadline: 26/04/2024 at 17:30

Extended Deadline: 28/04/2024 at 23:59

Marks: 200

1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your practical well before the deadline, as no extension will be granted.
- **The extended deadline has been put in place in case of loadshedding or other unforeseen circumstances. No further extension will be granted.**
- **Note that no tutor or lecturer will be available after the official deadline**
- You may not import any libraries that have not been imported for you, except `<iostream>`.
- You may `use namespace std;`
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire practical before you start coding.
- You will be afforded 20 upload opportunities per task.

- **You should no longer be using the online compiler. You should compile, run and test your code locally on your machine using terminal commands or makefiles**.
- **You have to use C++98 in order to get marks**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.**

3 Outcomes

Upon successful completion of this practical, students will be able to:

- Write a makefile that compiles a multi-file C++ project into an executable, understanding the relationship between header files, source files, and their dependencies.
- Use conditional statements effectively to manage complex decision-making processes within a software application.
- Compile and run C++ programs using terminal commands or makefiles, ensuring that the programs perform as expected under various conditions dictated by the provided specifications.

4 Structure

This practical consists of 2 tasks. Each task is self-contained and all of the code you will require to complete it will be provided in the appropriate Task folder. Each task will require you to submit a separate archive to an individual upload slot. That is, each separate task will require its own answer archive upload. You will upload Task 1 to Practical 5 Task 1 and so on. You can assume that all input will be valid (i.e. of the correct data type and within range)

5 Mark Distribution

Activity	**Mark**
Task 1 - Makefile	20
Task 2 - Conditional Statements	180
Total	**200**

Table 1: Mark Distribution

6 Task 1

Your task is to implement the makefile for this program.

6.1 Constants Header

The `Constants.h` file declares constant values used throughout the project.

```
#ifndef Constants_h
#define Constants_h

const float PI = 3.14;

#endif
```

6.2 Advanced Mathematics Functions

The `Advanced_Maths.h` file declares advanced mathematical functions.

```
#ifndef Advanced_Maths_h
#define Advanced_Maths_h

float squareRoot(float);
float areaOfCircle(float);
float areaOfTriangle(float, float);

#endif
```

6.2.1 Implementation Files

This section outlines the implementations provided in the corresponding CPP files.

```
#include "Advanced_Maths.h"
#include "Constants.h"
#include <math.h>

float squareRoot(float x){
    return sqrt(x);
}

float areaOfCircle(float radius){
    return PI * radius * radius;
}

float areaOfTriangle(float base, float height){
    return 0.5 * base * height;
}
```

6.3 Basic Mathematics Functions

The `Maths.h` file declares basic arithmetic operations.

```cpp
#ifndef Maths_h
#define Maths_h

float sum(float , float);
float subtract(float , float);
float multiply(float , float);
float divide(float , float);

#endif
```

6.3.1 Implementation Files

This section outlines the implementations provided in the corresponding CPP files.

```cpp
#include "Maths.h"

float sum(float x, float y){
    return x + y;
}

float subtract(float x, float y){
    return x - y;
}

float multiply(float x, float y){
    return x * y;
}

float divide(float x, float y){
    return x / y;
}
```

6.4 Calculator (Main)

The `Calculator.cpp` is the main function for the program.

```cpp
#include "Maths.h"
#include "Advanced_Maths.h"
#include <iostream>
using namespace std;

int main(){
    cout<< areaOfCircle(12.2) << endl;
    cout<< sum(2.3,3.7) << endl;
    return 0;
}
```

Instructions to construct the makefile Your makefile should:

1. Compile the object files Maths.o, Advanced_Maths.o and Calculator.o
2. Compile the executable and call the executable Prac5
3. Implement the run command such that it runs the executable
4. Implement the clean command to delete all .o files and the executable

When `make run` or `./Prac5` is executed, your code should compile and run the main function. Ensure that you compile with C++ 98.

You have been given all the files mentioned above in order to test your makefile.

7 Task 2

After a whirlwind of assessments, your skills in coding and problem-solving have not gone unnoticed. You're invited by a cryptic, environmentally-encoded message to join the elite "Climate Vanguard Guild." Your first mission: infiltrate the EcoDome, a bio-engineered habitat controlled by the shady TerraCorp. They're accused of manipulating environmental conditions for profit, threatening wildlife and ecosystems. Inside the EcoDome's servers are secret controls used to create illegal weather patterns for geopolitical advantage.

With your expertise, you're charged with adjusting the EcoDome's environmental systems to avert a looming ecological disaster and uncover TerraCorp's schemes. These systems, critical for temperature, pressure, and humidity adjustments, maintain stability and support thousands of species. Armed with knowledge and tools, you start tweaking these controls.

7.1 Header and Source File Creation Guide

This guide outlines the structure and responsibilities of the header ('.h') and source ('.cpp') files for the monitoring system. Each header file contains the declarations of functions related to the environmental controls for temperature, pressure, and humidity.

Detailed instructions on function implementations follow in 8.2.

By the end of 8.1 you should have created the following header ('.h') files:

- MonitoringSystem.h
- HumidityControl.h
- PressureControl.h
- TemperatureControl.h

7.1.1 Humidity Control Header and Implementation

The `HumidityControl.h` header file contains the function declarations related to humidity control.

```
#ifndef HUMIDITY_CONTROL_H
#define HUMIDITY_CONTROL_H

#include <iostream>

void adjustHumidity(float humidity, float temperature, float pressure);
void handleHumidAndHot(float humidity, float temperature);
void handleVeryHumid(float humidity);
void handleDryAndLowPressure(float humidity, float pressure);
void handleDry(float humidity);
void handleModerateHumidityWarm(float temperature);
void handleModerateHumidityCold(float temperature);
void handleHighPressureHumidity(float pressure);
void handleLowPressureHumidity(float pressure);
void handleStableConditionsHumidity();

#endif // HUMIDITY_CONTROL_H
```

In `HumidityControl.cpp`, include the `HumidityControl.h` header at the top:

```
#include "HumidityControl.h"

// Implement the functions to adjust humidity
```

7.1.2 Pressure Control Header and Implementation

The `PressureControl.h` file includes the necessary function declarations for pressure adjustment mechanisms.

```
#ifndef PRESSURE_CONTROL_H
#define PRESSURE_CONTROL_H

#include <iostream>

void adjustPressure(float pressure, float temperature, float humidity);
void handleHighPressureHighTemp(float pressure, float temperature);
void handleHighPressurePressure(float pressure);
void handleLowPressureDry(float pressure, float humidity);
void handleLowPressurePressure(float pressure);
void handleModeratePressureWarm(float temperature);
void handleModeratePressureHumid(float humidity);
void handleStableConditionsPressure();

#endif // PRESSURE_CONTROL_H
```

The corresponding source file `PressureControl.cpp` should start with the include directive for its header:

```
#include "PressureControl.h"

// Implement pressure adjustment functions
```

7.1.3 Temperature Control Header and Implementation

The `TemperatureControl.h` file declares functions related to temperature adjustments.

```
#ifndef TEMPERATURE_CONTROL_H
#define TEMPERATURE_CONTROL_H

#include <iostream>

void adjustTemperature(float temperature, float humidity, float pressure);
void handleHighTempHighPressureHighHumidity(float temperature);
void handleHighTempHighPressure(float temperature, float pressure);
void handleHighTempLowHumidity(float temperature, float humidity);
void handleHighTemp(float temperature);
void handleColdHighHumidity(float temperature, float humidity);
void handleCold(float temperature);
void handleStableConditionsTemperature();

#endif // TEMPERATURE_CONTROL_H
```

Then, in `TemperatureControl.cpp`, include its header file:

```
#include "TemperatureControl.h"

// Function implementations for temperature control
```

7.1.4 Monitoring System Header and Implementation

The header file `MonitoringSystem.h` includes the function declaration for the overall monitoring and adjustment process.

```
#ifndef MONITORING_SYSTEM_H
#define MONITORING_SYSTEM_H
```

```
#include "PressureControl.h"
#include "HumidityControl.h"
#include "TemperatureControl.h"
#include <iostream>

void monitorAndAdjust(float temperature, float pressure, float humidity);

#endif // MONITORING_SYSTEM_H
```

In `MonitoringSystem.cpp`, include the `MonitoringSystem.h` header and implement the `monitorAndAdjust` function:

```
#include "MonitoringSystem.h"

// Function implementations here
```

7.2 Function Implementation Specifications

Each '.cpp' file should implement the functions declared in its respective '.h' file. Pay attention to ensure that each '.cpp' file includes its corresponding header file to provide function prototypes, enabling proper compilation. Follow the instructions below carefully to implement the required functions.

7.2.1 Function Implementation Specifications for Humidity Adjustment

The humidity adjustment process requires assessing the levels of humidity, temperature, and pressure to engage the correct handling function.

- `void handleHumidAndHot(float humidity, float temperature)`
  This function receives humidity and temperature.

  1. Print "Humid and hot. Dehumidifying and cooling.\n"
  2. Decrease humidity by 20
  3. Decrease temperature by 5
  4. Print "Temperature set to: " + temperature + ".\n";
  5. Print "Humidity set to: " + humidity + ".\n";

- `void handleVeryHumid(float humidity)`
  This function receives humidity.

  1. Print "Very humid. Dehumidifying.\n"
  2. Decrease humidity by 15
  3. Print "Humidity set to: " + humidity + ".\n";

- `void handleDryAndLowPressure(float humidity, float pressure)`
  This function receives humidity and pressure.
  1. Print "Dry and low pressure. Moisturizing and pressurizing.\n"
  2. Increase humidity by 20
  3. Increase pressure by 25
  4. Print "Pressure set to: " + pressure + ".\n";
  5. Print "Humidity set to: " + humidity + ".\n";
- `void handleDry(float humidity)`
  This function receives humidity.
  1. Print "Dry only. Moisturizing.\n"
  2. Increase humidity by 10
  3. Print "Humidity set to: " + humidity + ".\n";
- `void handleModerateHumidityWarm(float temperature)`
  This function receives temperature.
  1. Print "Moderate humidity and warm. Slight cooling.\n"
  2. Decrease temperature by 3
  3. Print "Temperature set to: " + temperature + ".\n";
- `void handleModerateHumidityCold(float temperature)`
  This function receives temperature.
  1. Print "Moderate humidity and cold. Slight heating.\n"
  2. Increase temperature by 3
  3. Print "Temperature set to: " + temperature + ".\n";
- `void handleHighPressureHumidity(float pressure)`
  This function receives pressure.
  1. Print "High pressure. Depressurizing.\n"
  2. Decrease pressure by 15
  3. Print "Pressure set to: " + pressure + ".\n";
- `void handleLowPressureHumidity(float pressure)`
  This function receives pressure.
  1. Print "Low pressure. Pressurizing.\n"
  2. Increase pressure by 15
  3. Print "Pressure set to: " + pressure + ".\n";

- `void handleStableConditionsHumidity()`
  This function does not receive any parameters.

  1. Print "Humidity is stable. No action required.\n"

- `void adjustHumidity(float humidity, float temperature, float pressure)`
  called with current readings of humidity, temperature, and pressure. The function selects the appropriate action based on the humidity level:

  1. Print "Stabilizing humidity.\n"
  2. For `humidity > 75`:
     - If `temperature > 35`, call `handleHumidAndHot`.
     - Otherwise, call `handleVeryHumid`.
  3. For `humidity < 25`:
     - If `pressure < 950`, call `handleDryAndLowPressure`.
     - Otherwise, call `handleDry`.
  4. If `humidity` is between 25 and 75:
     - Based on the readings of `temperature` and `pressure`, in this order, invoke:
       - `handleModerateHumidityWarm` if `temperature > 25`.
       - else `handleModerateHumidityCold` if `temperature < 15`.
       - else `handleHighPressureHumidity` if `pressure > 1020`.
       - else `handleLowPressureHumidity` if `pressure < 980`.
       - else `handleStableConditionsHumidity` if none of the above conditions apply.

7.2.2 Function Implementation Specifications for Pressure Adjustment

The pressure adjustment process involves evaluating the current pressure, temperature, and humidity levels to determine which adjustment function to call. These functions are responsible for printing specific messages and "adjusting" the environmental variables accordingly.

- `void handleHighPressureHighTemp(float pressure, float temperature)` This function receives pressure and temperature.

  1. Print "High pressure and high temperature. Depressurizing and cooling.\n"
  2. Decrease pressure by 40
  3. Decrease temperature by 7
  4. Print "Pressure set to: " + pressure + ".\n";
  5. Print "Temperature set to: " + temperature + ".\n";

- `void handleHighPressurePressure(float pressure)`
  This function receives pressure.
  1. Print "High pressure. Depressurizing.\n"
  2. Decrease pressure by 30
  3. Print "Pressure set to: " + pressure + ".\n";
- `void handleLowPressureDry(float pressure, float humidity)`
  This function receives pressure and humidity.
  1. Print "Low pressure and dry. Pressurizing and moisturizing.\n"
  2. Increase pressure by 35
  3. Increase humidity by 15
  4. Print "Pressure set to: " + pressure + ".\n";
  5. Print "Humidity set to: " + humidity + ".\n";
- `void handleLowPressurePressure(float pressure)`
  This function receives pressure.
  1. Print "Low pressure. Pressurizing.\n"
  2. Increase pressure by 25
  3. Print "Pressure set to: " + pressure + ".\n";
- `void handleModeratePressureWarm(float temperature)`
  This function receives temperature.
  1. Print "Moderate pressure and warm. Minor cooling.\n"
  2. Decrease temperature by 2
  3. Print "Temperature set to: " + temperature + ".\n";
- `void handleModeratePressureHumid(float humidity)`
  This function receives humidity.
  1. Print "Moderate pressure and humid. Minor dehumidification.\n"
  2. Decrease humidity by 5
  3. Print "Humidity set to: " + humidity + ".\n";
- `void handleStableConditionsPressure()`
  This function does not receive any parameters.
  1. Print "Pressure is stable. No action required.\n"
- `void adjustPressure(float pressure, float temperature, float humidity)`
  This function receives pressure, temperature, and humidity, and calls other functions based on conditions.

1. Print "Stabilizing pressure.\n"
2. For pressure > 1200:
    - If temperature is greater than 38, call `handleHighPressureHighTemp(pressure, temperature)`
    - Otherwise, call `handleHighPressurePressure(pressure)`
3. For pressure < 800:
    - If humidity is less than 30, call `handleLowPressureDry(pressure, humidity)`
    - Otherwise, call `handleLowPressurePressure(pressure)`
4. Otherwise:
    - If temperature is greater than 25, call `handleModeratePressureWarm(temperature)`
    - Else If humidity is greater than 50, call `handleModeratePressureHumid(humidity)`
    - Otherwise, call `handleStableConditionsPressure()`

7.2.3 Function Implementation Specifications for Temperature Adjustment

The temperature adjustment process involves calling specific functions based on temperature, humidity, and pressure readings. Each function must print a designated message and adjust the variables as required.

- `void handleHighTempHighPressureHighHumidity(float temperature)` This function receives temperature.
    1. Print "Critical: High temp, high pressure, high humidity. Emergency shutdown.\n"
    2. Decrease temperature by 20
    3. Print "Temperature set to: " + temperature + ".\n";
- `void handleHighTempHighPressure(float temperature, float pressure)`
  This function receives temperature and pressure.
    1. Print "High temp and pressure. Cooling and depressurizing.\n"
    2. Decrease temperature by 15
    3. Decrease pressure by 30
    4. Print "Pressure set to: " + pressure + ".\n";
    5. Print "Temperature set to: " + temperature + ".\n";

- `void handleHighTempLowHumidity(float temperature, float humidity)`
  This function receives temperature and humidity.
  1. Print "High temp, low humidity. Mild cooling and moisturizing.\n"
  2. Decrease temperature by 10
  3. Increase humidity by 15
  4. Print "Humidity set to: " + humidity + ".\n";
  5. Print "Temperature set to: " + temperature + ".\n";
- `void handleHighTemp(float temperature)`
  This function receives temperature.
  1. Print "High temp only. Standard cooling.\n"
  2. Decrease temperature by 5
  3. Print "Temperature set to: " + temperature + ".\n";
- `void handleColdHighHumidity(float temperature, float humidity)`
  This function receives temperature and humidity.
  1. Print "Cold and humid. Heating and slight dehumidification.\n"
  2. Increase temperature by 10
  3. Decrease humidity by 10
  4. Print "Humidity set to: " + humidity + ".\n";
  5. Print "Temperature set to: " + temperature + ".\n";
- `void handleCold(float temperature)`
  This function receives temperature.
  1. Print "Cold only. Heating.\n"
  2. Increase temperature by 5
  3. Print "Temperature set to: " + temperature + ".\n";
- `void handleStableConditionsTemperature()`
  This function does not receive any parameters.
  1. Print "Temperature is stable. No action required.\n"
- `void adjustTemperature(float temperature, float humidity, float pressure)`
  This function receives temperature, humidity, and pressure.
  1. Print "Stabilizing temperature.\n"
  2. If Temperature > 40:
     - If pressure is greater than 1100 and humidity greater than 60, call `handleHighTempHighPressureHighHumidity(temperature)`

  - If pressure is greater than 1100, but humidity id not greater than 60, call `handleHighTempHighPressure(temperature, pressure)`
  - If pressure less than or equal to 1100 and humidity less than 30, call `handleHighTempLowHumidity(temperature, humidity)`
  - Otherwise if pressure is less than or equal to 1100 but humidity is more than or equal to 30, call
    `handleHighTemp(temperature)`

3. If temperature < 10:
   - If humidity is greater than 70, call `handleColdHighHumidity(temperature, humidity)`
   - Otherwise, call `handleCold(temperature)`
4. Otherwise, call `handleStableConditionsTemperature()`

7.2.4 Function Implementation Specifications for the Monitoring System

The `monitorAndAdjust` function is the coordinator for applying environmental adjustments. This function is responsible for one cycle of adjustments using the current readings of temperature, pressure, and humidity.

- `void monitorAndAdjust(float temperature, float pressure, float humidity)` This function receives temperature, pressure, and humidity as parameters and orchestrates the environmental adjustments through three specific adjustment functions.
  1. Call `adjustHumidity(humidity, temperature, pressure)`:
     - Adjusts the humidity levels based on the current temperature and pressure readings.
  2. Call `adjustPressure(pressure, temperature, humidity)`:
     - Modifies the atmospheric pressure in response to the existing temperature and humidity conditions.
  3. Call `adjustTemperature(temperature, humidity, pressure)`:
     - Adjusts the temperature based on the prevailing humidity and pressure readings.

8 Submission checklist

For Task 1:

- Archive (zip) all the files used for Task 1 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - Makefile
- Upload the archive to FitchFork Practical 5 Task 1 before the deadline

For Task 2:

- Archive (zip) all files used for Task 2 and rename the archive uXXXXXXXX.zip where XXXXXXX is your student number. The zip should include:
  - `MonitoringSystem.h`
  - `MonitoringSystem.cpp`
  - `HumidityControl.h`
  - `HumidityControl.cpp`
  - `PressureControl.h`
  - `PressureControl.cpp`
  - `TemperatureControl.h`
  - `TemperatureControl.cpp`
- Upload the archive to FitchFork Practical 5 Task 2 before the deadline