PPL LAB 6

SAHIL BONDRE: U18CO021

1. Declare a class called author having author_name as private data member. Extend author class to have two sub classes called book_publication & paper_publication. Each of these classes have private member called title. Show usage of dynamic method dispatch (dynamic polymorphism) to display book or paper publications of a given author. Use command line arguments for inputting data.

```
#include <iostream>

#include <string>


using namespace std;


class author {

 private:

  string author_name;

  string title;


 public:

  author(string name, string title) : author_name(name){};


  virtual void print() {

    cout << "Called in Base Class\n";

    cout << "Author: " << author_name << ", Title: " << title << "\n";

  }

};
```

```
class book_publication : public author {
 public:
  book_publication(string name, string title) : author(name, title){};

  void print() { cout << "Called in Book Class\n"; }
};

class paper_publication : public author {
 public:
  paper_publication(string name, string title) : author(name, title){};
  void print() { cout << "Called in Paper Class\n"; }
};

int main(int argc, char **argv) {
  if (argc != 2) {
    cout << "Usage: ./a.out <author-name>\n";
    return 0;
  }

  string query = *(argv + 1);

  author *a = new paper_publication("hawking", "Universe");
  author *b = new book_publication("torvalds", "Linux");

  if (query == "hawking") {
    a->print();
```

```
  } else if (query == "torvalds") {

    b->print();

  } else {

    cout << "Error: author not found\n";

  }



  return 0;

}
```

```
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06> g++ .\q1.cpp
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06> .\a.exe
Usage: ./a.out <author-name>
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06> .\a.exe torvalds
Called in Book Class
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06>
```

2. Write a class named Rectangle to represent a rectangle. It contains the following members:
   Data: width (double) and height (double) that specify the width and height of the rectangle.
   Methods:
   1. A no-arg constructor that creates a default rectangle.
   2. A constructor that creates a rectangle with the specified width and height.
   3. A method named getArea() that returns the area of this rectangle.
   4. A method named getPerimeter() that returns the perimeter

```
#include <iostream>

#include <string>



using namespace std;



class rectangle {

  private:

    double width;

    double height;
```

```
  public:
    rectangle(): width(0), height(0) {};
    rectangle(double w, double h): width(w), height(h) {};

    double getArea() {
      return width * height;
    }

    double getPerimeter() {
      return 2 * (width + height);
    }
};


int main() {
  rectangle a(4, 5);
  rectangle b;

  cout << a.getArea() << "\n";
  cout << b.getArea() << "\n";

  return 0;
}
```

```
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06> g++ .\q2.cpp
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06> .\a.exe
20
0
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06>
```

3. It is required to compute SPI (semester performance index) of n students of a class for their registered subjects in a semester.
   Assume that all students register for 6 subjects and each subject carry 5 credits. Also, follow SVNIT convention and method for computation of SPI.
   Declare a class called student having following data members:
   id_no, grades_obtained and SPI.
   Define constructor, display and calculate_spi methods. Define main to process data of n students.

```cpp
#include <iostream>

#include <string>

#include <vector>


using namespace std;


class student {

  private:

    vector<double> grades;

    int id;

    double spi;

  public:

    student(vector<double> &v, int id): id(id), grades(v) {};


    void calculate_spi() {

      double sum = 0;
```

```
      for(auto i: grades) {
        sum += i;
      }

      spi = sum / (double) grades.size();
    }

    void display() {
      calculate_spi();
      cout << "ID: " << id << ", spi: " << spi << "\n";
    }

};

int main() {
  int n;
  cout << "Enter n: ";
  cin >> n;
  for(int i = 0; i < n; ++i) {
    cout << "Student ID: " << i + 1 << "\nEnter grades: ";
    vector<double> v(6);
    for(int j = 0; i < 6; ++i) {
      cin >> v[i];
    }

    student s(v, i + 1);
```

```
    s.display();

  }



  return 0;

}
```

```
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06> .\a.exe
Enter n: 1
Student ID: 1
Enter grades: 7.8 8.2 9.8 8.0 7.8
8.9
ID: 7, spi: 8.41667
PS F:\code\github.com\godcrampy\college-notes\ppl\lab-06>
```

4. It is required to maintain and process the status of total 9 resources. The status value is to be stored in an integer array of dimensions 3x3. The valid status of a resource can be one of the following:
   free: indicated by integer value 0
   occupied: indicated by integer value 1
   inaccessible: indicated by integer value 2
   Declare a class called ResourcesStatus, having data member called statusRef, referring to a two dimensional array (3x3) of integers to be used to refer to the above mentioned status values. Define a member method called processStausCount that counts and displays total number of free resources, total number of occupied resources and total number of inaccessible resources. The exception to be raised and handled if total number of occupied resources exceeds total number of free resources. The handler marks status of all inaccessible resources as free. Accept initial status values from command line arguments and initialize the array. Raise and handle user defined exception if invalid status value given.

```
#include <cstdlib>

#include <iostream>

#include <vector>



using namespace std;



class ResourcesStatus {

 public:
```

```
vector<vector<int>> processStatus;


ResourcesStatus() {
  vector<vector<int>> v(3, vector<int>(3, 0));
  processStatus = v;
}
void processStatusCount(void) {
  int free = 0, occupied = 0, inaccessible = 0;
  for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
      if (processStatus[i][j] == 0) {
        ++free;
      } else if (processStatus[i][j] == 1) {
        ++occupied;
      } else if (processStatus[i][j] == 2) {
        ++inaccessible;
      }
    }
  }
  cout << "Number of free processes: " << free << endl;
  cout << "Number of occupied processes: " << occupied << endl;
  cout << "Number of inaccessible processes: " << inaccessible << endl;
  try {
    if (occupied <= free) {
      cout << "Process count is ideal. No exception to raise" << endl;
    } else {
```

```
        throw "Occupied resources have exceeded free resource";
      }
    } catch (const char* msg) {
      cout << msg << "\n";
      for (int i = 0; i < 3; ++i) {
        for (int j = 0; j < 3; ++j) {
          if (processStatus[i][j] == 2) {
            processStatus[i][j] = 0;
          }
        }
      }
    }
  }

  void displayProcessStatus() {
    cout << "Process status in 3x3 array: " << endl;
    for (int i = 0; i < 3; ++i) {
      for (int j = 0; j < 3; ++j) {
        cout << processStatus[i][j] << " ";
      }
      cout << endl;
    }
  }
};

int main(int argc, char* argv[]) {
```

```
  if (argc != 10) {
    cout << "Usage: ./e.out [9 statuses]\n";
    return 1;
  }
  int row = 0, col = 0;
  ResourcesStatus* rs = new ResourcesStatus();
  for (int i = 1; i < argc; ++i) {
    int status = atoi(argv[i]);
    if (status != 0 && status != 1 && status != 2) {
      cout << "status must always be 0, 1 or 2\n";
      return -1;
    }
    if (col == 3) {
      ++row;
      col = 0;
    }
    rs->processStatus[row][col] = status;
    ++col;
  }
  rs->displayProcessStatus();
  rs->processStatusCount();
  rs->displayProcessStatus();
  return 0;
}
```