ITE1015 창의적 소프트웨어 프로그래밍

2024 가을학기 중간고사

공지

1. 시작하기 전 반드시 페이지 수를 확인하세요 (표지 제외 총 10쪽, 양면인쇄, 100점 만점).
2. 답안지 각 페이지마다 반드시 학번, 이름, 쪽번호를 적으세요.
3. 퇴실은 시험 시작 1시간 후부터 가능합니다.
4. 문제에 특별한 제약사항이 없으면 C++03 버전의 모든 C++ 문법과 STL 사용이 가능합니다.

공백

1. 각 문항의 설명이 맞으면 O, 틀리면 X 로 답하시오. (총 10 점 각 1 점)

(1) 다형성(polymorphism)을 지원하는 클래스를 만들기 위해 가상생성자와 가상소멸자를 사용해야 한다. **X**

(2) 함수 기본 인자(default arguments)는 함수의 선언과 정의 양 쪽에 모두 작성하는 것이 좋다. **X**

(3) 파생 클래스(derived class)의 포인터 타입을 기본 클래스(base class)의 포인터타입으로 변환하는 것은 정의되지 않은 동작 (undefined behavior)를 만들지 않는다. **O**

(4) C++의 참조자(reference)는 항상 선언할 때 참조 대상을 지정해야 하며 선언 이후에 변경할 수 없다. **O**

(5) new로 할당한 메모리는 메모리 누수(memory leak)를 방지하기 위해 delete나 free로 해제해야 한다. **X**

(6) 사용자가 기본 생성자를 작성하지 않은 경우 컴파일러가 기본 생성자를 만든다. **O**

(7) delete p; 로 할당된 메모리를 해제한 후에 p는 널 포인터 값을 갖는다. **X**

(8) C++에서 서브타입 다형성(subtype polymorphism)을 사용하기 위해선 객체의 포인터 혹은 참조자를 사용해야 한다. **O**

(9) 함수 오버로딩(function overloading)을 사용해서 함수 이름과 파라메터 순서 및 타입이 같고 반환값 타입만 다른 서로 다른 함수를 정의할 수 있다. **X**

(10) 함수 기본 인자(default arguments)를 사용할 때 어떤 인자가 기본 인자로 선언되었으면 해당 인자의 뒤에 나오는 모든 인자는 기본 인자이다. **O**

- **Solution**

(1) X (2) X (3) O (4) O (5) X (6) O (7) X (8) O (9) X (10) O

2. 다음 C++ 코드를 보고 실행 결과를 작성하시오. 만약 컴파일 에러와 런타임 에러가 발생한다면 해당 에러가 왜 발생하는지 그 이유를 간단히 설명하시오. (총 20점) (각 소문항별로 부분점수 없음 (0점 아니면 5점))

(1) (5 점)

```
#include <iostream>
using namespace std;
class B {
public:
    B() { cout << "B ctor()" << endl; }

    ~B() { cout << "B dtor()" << endl; }
};
class D1 : public B {
public:
    D1() { cout << "D1 ctor()" << endl; }

    ~D1() { cout << "D1 dtor()" << endl; }
};
class D2 : public D1 {
public:
    D2() { cout << "D2 ctor()" << endl; }

    ~D2() { cout << "D2 dtor()" << endl; }
};
int main() {
    cout << "main() begins" << endl;
    D2 obj;
    cout << "main() ends" << endl;
    return 0;
}
```

- **Solution**

```
main() begins
B ctor()
D1 ctor()
D2 ctor()
main() ends
D2 dtor()
D1 dtor()
B dtor()
```

(2) (5 점)

```
#include <iostream>
using namespace std;

void print(const int a, int b = 5) {
    cout << "Two integers: " << a << " and " << b << endl;
}

void print(int a) {
    cout << "One integer: " << a << endl;
}

int main() {
    print(10);
    return 0;
}
```

- **Solution**

컴파일 에러: 컴파일 시간에 main() 함수의 print(10) 이 어떤 print()를 호출해야 하는지 컴파일러가 결정할 수 없다. (Ambiguous call)

(3) (5 점)

```
#include <iostream>
#include <vector>
using namespace std;
class Data {
public:
    int value;
    Data(int v) : value(v) {}
    void print() const {cout << "Val: " << value << endl; }
};
void printVector(const vector<Data>& vec, int index) {
    if (index >= 0) { vec.at(index).print(); }
    else { vec[index].print(); }
}
int calculateIndex(int base, int offset) {
    if (base % 2 == 0) {
        return base + offset;
    } else {
        return -(base + offset);
    }
}
int main() {
    vector<Data> vec;

    for (int i = 1; i <= 5; ++i) {
        vec.push_back(Data(i * 10));
    }

    int baseIndex = 3;
    int offset = 2;

    int index = calculateIndex(baseIndex, offset);

    printVector(vec, index);

    return 0;
}
```

- Solution

  런타임 에러: vec 벡터를 접근하기 위한 index가 -5로 되어 있어서 잘못된 원소를 접근함.

(4) (5점)

```
#include <map>
#include <iostream>
using namespace std;
int main(void) {
    map <string, double> m;
    for (int i=0; i<4; i++)
        m.insert(make_pair("string" + to_string(i), 0.5*i));
    map<string, double>::iterator it;
    for (it = m.begin(); it != m.end(); ++it) {
        cout << " " << it->first << "," << it->second << endl;
    }
    m.insert(make_pair("apple", 10));
    m["orange"] = 3.14;
    m["string0"] = 111;
    map<string, double>::reverse_iterator rit;
    for (rit = m.rbegin(); rit != m.rend(); ++rit) {
        cout << " " << rit->first;
        cout << "," << rit->second << endl;
    }
    it = m.find("apple");
    cout << "output " << it->first;
    cout << " " << (*it).second << endl;
    m.clear();
    return 0;
}
```

- Solution

```
 string0,0
 string1,0.5
 string2,1
 string3,1.5
 string3,1.5
 string2,1
 string1,0.5
 string0,111
 orange,3.14
 apple,10
output apple 10
```

3. C++의 네가지 다형성(polymorphism)에 대해서 각 타입의 다형성이 C++의 어떤 기능에 대응하는지를 포함하여 서술하시오. (10점)

- Solution

서브타입 다형성 (Subtype polymorphism)
Ability to access a derived class object through its base class interface
Often simply referred to as just "polymorphism".
기반 클래스 인터페이스를 통해 파생 클래스 객체에 접근할 수 있는 능력
종종 단순히 "다형성"이라고 불립니다.

애드혹 다형성 (Ad hoc polymorphism)
Allows functions with the same name to work differently for each type
Overloading in C++
동일한 이름의 함수가 각 타입에 대해 다르게 동작할 수 있도록 합니다
C++에서의 오버로딩

파라메트릭 다형성 (Parametric polymorphism)
Allows a function or a data type to be written generically
Templates in C++
함수나 데이터 타입이 일반적으로 작성될 수 있도록 허용합니다
C++에서의 템플릿

강제 다형성 (Coercion polymorphism)
(Implicit or explicit) casting in C++
C++에서의 암시적 또는 명시적 형 변환

- Criteria
  - 각 polymorphism 이름당 (C++ 기능이 맞으면) **+1점**
  - 각 polymorphism 설명당 (설명이 맞으면) **+1.5점**

4. 주어진 문자열에서 특정 문자열을 찾아 다른 문자열로 교체하는 C++ 함수를 구현하시오. (15 points)

함수의 타입: `std::string replace(const std::string &str, const std::string &target, const std::string &replacement)`

함수 입력:

- str: 원본 문자열
- target: 찾아서 교체할 문자열
- replacement: 교체할 문자열

함수 출력: 주어진 str에서 target 문자열을 모두 replacement로 바꾼 새로운 문자열을 반환.

예시

`replace("The cat sat on the mat with the cat.", "cat", "dog")` -> return `"The dog sat on the mat with the dog."`

`replace("ABCD", "EF", "GH")` -> return `"ABCD"`

- Solution

```
std::string replace(const std::string &str, const std::string &target, const std::string &replacement) {
    std::string result = str;
    size_t pos = 0;

    while ((pos=result.find(target,pos)) != std::string::npos) {
        result.replace(pos, target.length(), replacement);
        pos += replacement.length();
    }
    return result;
}
```

- Criteria
    - 함수가 정확하게 정의됨 **+15점**
        - Given string에서 target을 최소 하나라도 찾음 +5점
        - 최소 하나의 target이 변경됨 +5점
            - 모든 target이 변경됨 +5점
        - 부적절한 예외처리로 일부 입력에서 런타임 오류발생 **-5점**

5. 다음 기능 설명을 읽고 여러 도형을 그릴 수 있는 C++ 프로그램을 작성하시오. (45점)

a. 캔버스(Canvas) 기능 (10점)

(1) 캔버스 생성: 사용자로부터 가로 및 세로 길이를 입력받아 캔버스를 생성한다.

(2) 캔버스 크기 재설정: 사용자로부터 가로 및 세로 길이를 입력받아 캔버스의 크기를 다시 설정한다. 기존에 생성한 캔버스보다 가로 혹은 세로 길이가 더 짧은 경우 캔버스의 가로 혹은 세로 길이가 축소되고 축소된 영역 밖에 그려진 내용은 사라진다. 기존에 생성한 캔버스보다 가로 혹은 세로 길이가 더 긴 경우 가로 혹은 세로 길이를 늘린다. 기존에 생성한 캔버스가 없는 경우 사용자 입력 길이로 캔버스를 생성한다.

(3) 한 점 그리기: 사용자로부터 가로, 세로 좌표와 점에 그릴 문자를 입력받고 해당 좌표에 주어진 문자를 그린다. 이 때 빈 공간은 '.' 으로 표시한다.

(4) 캔버스 출력: 캔버스에 그려진 내용을 화면에 출력한다.

(5) 모두 지우기: 캔버스에 그려진 내용을 전부 지우고 빈 공간으로 만든다.

b. 도형 기능 (20점)

(1) 도형 생성: 사용자가 그리고자 하는 도형을 생성한다. 도형은 `Shape` 클래스의 파생 클래스로 구현되어야 하며, 다음 네 종류의 도형들을 지원해야 한다.

(a) 사각형(Rectangle): 좌상단 좌표, 너비, 높이, 문자 문자를 입력받아 생성.

(b) 마름모(Diamond): 중심 좌표, 중심부터의 거리, 문자 문자를 입력받아 생성.

(c) 삼각형(Upward Triangle): 중심 좌표, 높이를 입력받아 생성.

(d) 역삼각형(Downward Triangle): 중심 좌표, 높이를 입력받아 생성.

(2) 도형 그리기: 캔버스에 해당 도형을 그린다. 기존에 캔버스에 그린 도형과 영역이 겹치는 경우 나중에 그린 도형으로 덮어쓴다. 도형의 일부 혹은 전부가 현재 캔버스의 영역을 벗어날 경우 캔버스 영역 안에 있는 부분만 그린다.

c. 사용자 명령어 처리 기능: 사용자는 아래 6종류의 명령어를 사용해서 도형을 생성하고 캔버스에 그림을 그려서 출력할 수 있다. (15점)

(1) `add` [도형] [매개변수]: 새로운 도형을 생성하여 캔버스에 추가한다. 각 도형 별 필요 매개변수는 아래와 같다. 매개변수가 도형을 그리기 적합하지 않은 경우(예: x, y 좌표나 너비, 높이, 중심부터 거리 등이 음수인 경우, 지원하는 도형이 아닌 경우 등) 해당 명령어는 무시한다.

(a) `add rect` [좌상단 x] [좌상단 y] [너비] [높이] [문자]

(b) `add diamond` [중심 x] [중심 y] [중심부터 거리] [문자]

(c) `add tri_up` [중심 x] [중심 y] [높이] [문자]

(d) `add tri_down` [중심 x] [중심 y] [높이] [문자]

(2) `delete` [도형 인덱스]: 해당 인덱스의 도형을 삭제한다. 인덱스가 유용하지 않은 경우 해당 명령어는 무시한다.

(3) `draw`: 캔버스에 추가된 모든 도형을 그리고 캔버스 내용을 화면에 출력한다.

(4) `dump`: 저장된 도형들의 정보를 출력한다.

(5) `resize` [새 너비] [새 높이]: 캔버스 크기를 재설정한다.

(6) `quit`: 프로그램을 종료한다.

위 기능을 구현하기 위해 `Canvas` 클래스와 `Shape` 클래스를 정의하였다. 캔버스 기능은 `Canvas` 클래스의 멤버 함수를 정의하여 구현해야 한다. 각 도형은 `Shape` 클래스를 상속받아 구현해야 한다. 필요시 일반 함수 및 `private` 멤버 함수를 추가할 수 있으나 `public` 멤버는 추가할 수 없다.

```
class Canvas {
public:
    Canvas(int row, int col):_row(row),_col(col){
        Clear();
    }
    ~Canvas() {}

    void Resize(size_t w, size_t h);

    void DrawPixel(int x, int y, char ch);

    void Print();

    void Clear();

private:
    int _row,_col;
    std::vector<std::vector<char>> _v;
};

class Shape {
public:
    Shape(int x, int y, char ch): _x(x),_y(y),_ch(ch){}
    virtual ~Shape() {}
    virtual void Draw(Canvas &canvas) {};

protected:
    int _x,_y;
    char _ch;
};
```

프로그램의 실행 예시는 아래와 같다. (굵은 글씨는 사용자 입력을 나타낸다.)

```
10 10⏎
 0123456789
0..........
1..........
2..........
3..........
4..........
5..........
6..........
7..........
8..........
9..........

add rect 4 4 3 3 *⏎
draw⏎
 0123456789
0..........
1..........
2..........
3..........
4....***...
5....***...
6....***...
7..........
8..........
9..........

add tri_down 3 3 3 @⏎
draw⏎
 0123456789
0..........
1.@@@@@....
2..@@@.....
3...@......
4....***...
5....***...
6....***...
7..........
8..........
9..........

add tri_up 5 7 3 #⏎
add diamond 2 5 2 ?⏎
draw⏎
 0123456789
0..........
1.@@@@@....
2..@@@.....
3...@......
4....***...
5..?.***...
6.???***...
7?????#....
8.???###...
9..?#####..
```

```
(continue)
add rect 5 5 8 4 +⏎
dump⏎
0 rect 4 4 3 3 *
1 tri_down 3 3 3 @
2 tri_up 7 7 3 #
3 diamond 2 5 2 ?
4 rect 5 5 8 4 +

draw⏎
 0123456789
0..........
1.@@@@@....
2..@@@.....
3...@......
4....***...
5..?.*+++++
6.???*+++++
7?????+++++
8.???#+++++
9..?#####..

delete 5⏎
delete 0⏎
dump⏎
0 tri_down 3 3 3 @
1 tri_up 7 7 3 #
2 diamond 2 5 2 ?
3 rect 5 5 8 4 +

draw⏎
 0123456789
0..........
1.@@@@@....
2..@@@.....
3...@......
4..........
5..?..+++++
6.???.+++++
7?????+++++
8.???#+++++
9..?#####..

resize 15 10⏎
draw⏎
 012345678901234
0...............
1.@@@@@.........
2..@@@..........
3...@...........
4...............
5..?..++++++++..
6.???.++++++++..
7?????++++++++..
8.???#++++++++..
9..?#####.......

quit⏎
```

- Solution

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

class Canvas;

class Shape {
public:
    Shape(int x, int y, char ch) : _x(x), _y(y), _ch(ch){}
    virtual ~Shape() {}
    virtual void Draw(Canvas &canvas) {};

protected:
    int _x, _y;
    char _ch;
};

class Canvas {
public:
    Canvas(int row, int col) : _row(row), _col(col) {
        Clear();
    }
    ~Canvas() {}

    void Resize(size_t w, size_t h) {
        _col = w;
        _row = h;
        Clear();
    }

    void DrawPixel(int x, int y, char ch) {
        if(x >= 0 && x < _col && y >= 0 && y < _row) {
            _v[y][x] = ch;
        }
    }

    void Print() {
        std::cout << " ";
        for(int col = 0; col < _col; ++col){
            std::cout << (col % 10);
        }
        std::cout << std::endl;
        for(int row = 0; row < _row; ++row){
            std::cout << (row % 10);
            for(int col = 0; col < _col; ++col){
                std::cout << _v[row][col];
            }
            std::cout << std::endl;
        }
    }

    void Clear() {
        _v.clear();
        _v.resize(_row, std::vector<char>(_col, '.'));
    }


private:
    int _row, _col;
    std::vector<std::vector<char>> _v;
};
```

```
class Rectangle : public Shape {
public:
    Rectangle(int x, int y, int width, int height, char brush)
        : Shape(x, y, brush), _width(width), _height(height) {}

    virtual void Draw(Canvas &canvas) override {
        for(int i = 0; i < _height; ++i){
            for(int j = 0; j < _width; ++j){
                canvas.DrawPixel(_x + j, _y + i, _ch);
            }
        }
    }
private:
    int _width, _height;
};

class UpTriangle : public Shape {
public:
    UpTriangle(int x, int y, int height, char brush)
        : Shape(x, y, brush), _height(height) {}

    void Draw(Canvas &canvas) override {
        for(int i = 0; i < _height; ++i){
            for(int j = -i; j <= i; ++j){
                canvas.DrawPixel(_x + j, _y + i, _ch);
            }
        }
    }
private:
    int _height;
};

class DownTriangle : public Shape {
public:
    DownTriangle(int x, int y, int height, char brush)
        : Shape(x, y, brush), _height(height) {}

    void Draw(Canvas &canvas) override {
        for(int i = 0; i < _height; ++i){
            for(int j = -i; j <= i; ++j){
                canvas.DrawPixel(_x + j, _y - i, _ch);
            }
        }
    }
private:
    int _height;
};

class Diamond : public Shape {
public:
    Diamond(int x, int y, int radius, char brush)
        : Shape(x, y, brush), _radius(radius) {}

    void Draw(Canvas &canvas) override {
        for(int i = 0; i <= _radius; ++i){
            for(int j = -i; j <= i; ++j){
                canvas.DrawPixel(_x + j, _y + i, _ch);
            }
        }
        for(int i = _radius - 1; i >= 0; --i){
            for(int j = -i; j <= i; ++j){
                canvas.DrawPixel(_x + j, _y + (2 * _radius - i), _ch);
            }
        }
    }
private:
    int _radius;
};
```

```cpp
int main(){
    int canvas_width, canvas_height;
    std::cin >> canvas_width >> canvas_height;
    Canvas canvas(canvas_height, canvas_width);
    canvas.Print();
    std::vector<Shape*> shapes;
    std::vector<std::string> list;
    std::string command, str;

    while(true){
        std::cin >> command;
        if(command == "add"){
            std::string shape_type;
            std::cin >> shape_type;
            if(shape_type == "rect"){
                int x, y, width, height;
                char brush;
                std::cin >> x >> y >> width >> height >> brush;
                shapes.push_back(new Rectangle(x, y, width, height, brush));
                str = shape_type + " " + std::to_string(x) + " " + std::to_string(y) + " " +
                      std::to_string(width) + " " + std::to_string(height) + " " + std::string(1, brush);
                list.push_back(str);
            }
            else if(shape_type == "tri_up"){
                int x, y, height;
                char brush;
                std::cin >> x >> y >> height >> brush;
                shapes.push_back(new UpTriangle(x, y, height, brush));
                str = shape_type + " " + std::to_string(x) + " " + std::to_string(y) + " " +
                     std::to_string(height) + " " + std::string(1, brush);
                list.push_back(str);
            }
            else if(shape_type == "tri_down"){
                int x, y, height;
                char brush;
                std::cin >> x >> y >> height >> brush;
                shapes.push_back(new DownTriangle(x, y, height, brush));
                str = shape_type + " " + std::to_string(x) + " " + std::to_string(y) + " " +
                      std::to_string(height) + " " + std::string(1, brush);
                list.push_back(str);
            }
            else if(shape_type == "diamond"){
                int x, y, radius;
                char brush;
                std::cin >> x >> y >> radius >> brush;
                shapes.push_back(new Diamond(x, y, radius, brush));
                str = shape_type + " " + std::to_string(x) + " " + std::to_string(y) + " " +
                      std::to_string(radius) + " " + std::string(1, brush);
                list.push_back(str);
            }
        }
        else if(command == "draw"){
            canvas.Clear();
            for(int i = 0; i < shapes.size(); ++i){
                shapes[i]->Draw(canvas);
            }
            canvas.Print();
        }
        else if(command == "delete"){
            int index;
            std::cin >> index;
            if(index >= 0 && index < shapes.size()){
                delete shapes[index];
                shapes.erase(shapes.begin() + index);
                list.erase(list.begin() + index);
            }
        }
        else if(command == "dump"){
            for(int i = 0; i < list.size(); i++){
                std::cout << i << ": " << list[i] << std::endl;
            }
        }
        else if(command == "resize"){
            int new_width, new_height;
            std::cin >> new_width >> new_height;
            canvas.Resize(new_width, new_height);
        }
        else if(command == "quit"){
            break;
        }
    }
    for(int i = 0; i < shapes.size(); ++i){
        delete shapes[i];
    }
    return 0;
}
```

- Criteria **(Solution은 참고용)**
  - 캔버스(Canvas) 기능 (함수의 구현만 중점적으로 확인) **(10점)**
    - (constructor) 캔버스가 정상적으로 생성됨. **+2점**
    - (resize) 캔버스 크기 재설정 가능. **+2점**
    - (drawpixel) 캔버스에 주어진 좌표의 문자가 해당하는 brush 문자로 정상적으로 변경됨. **+2점**
    - (print) 캔버스가 정상적으로 출력 됨. **+2점**
      - Index 미표시. **-1점**
    - (clear) 캔버스가 정상적으로 초기화 됨. **+2점**
  - 도형(Shape) 기능 **(20점)**
    - 각 도형에서 필요한 private member 변수가 선언됨. **각+1점 (4점)**
    - 각 도형의 constructor가 정확하게 정의됨. **각+1점 (4점)**
    - 각 도형에서 draw 함수가 override됨. **각+1점 (4점)**
      - 해당 draw 함수가 해당 도형에 맞게 정의됨. **각+2점 (8점)**
  - 명령어 & main() **(15점)**
    - 최초 캔버스 생성 시 미출력. **-1점**
    - (add, delete, resize)각 명령어가 정확한 arg 개수를 받음. **각+1점 (3점)**
    - (add) 각 도형 arg마다 해당되는 정확한 class (Rectangle, UpTriangle..)가 생성, 추가 및 저장됨. **각+1점 (4점)**
    - (delete) 인덱스 arg의 도형이 정상적으로 삭제됨. **+1점**
    - (draw) 캔버스와 도형들이 정상적으로 출력됨. **+2점**
      - 일부 도형만 출력되는 경우. **-1점**
      - 도형이 완전하게 출력되지 않은 경우. **-1점**
    - (dump) 저장된 도형들의 정보가 정상적으로 출력됨. **+2점**
      - 일부만 출력되는 경우. **-1점**
    - (resize) 캔버스의 크기가 정상적으로 재설정됨. **+1점**
    - Memory free가 정상적으로 됨. (memory leak이 없음) **+1점**
    - Runtime 오류가 정상적으로 handle됨. (invalid index, undefined command 등) **+1점**