CS 246 MIDTERM REVIEW

1. Streams (10 mins)

You are strolling down Ring Road when, suddenly, Greek goddess Athena appears out of the bushes to gift you, her valiant hero(ine), the ancient scroll of legend midterm_answer_key.txt. She herself has been unable to read this prophetic text for 4,000 years, and now with the modern power of computing, it is up to you to read the fabled passage that contains all the world's CS midterm answers within it. Word on Mount Olympus is that the number of words in each line corresponds to an ASCII encoding of a character, and that the characters put together form the all-important passage. Write your program in C++ to save CS students around the world.

```cpp
int main() {
    ifstream file{"midterm_answer_key.txt"};
    string line;
    while (getline(file, line)) {
        string word;
        int counter = 0;
        istringstream iss{line};
        while (iss >> word) ++counter;
        char decrypt_char = counter;
        cout << decrypt_char;
    }
}
```

2. References (5 mins)

While you are mindlessly scrolling TikTok, you suddenly become absolutely engrossed by the various nuances of references in C++. It is frighteningly gripping, to the extent that you begin question your C++ programming ability. In a desperate attempt to hold onto your sanity, you begin to frantically write out the peculiarities of references that plague your mind. Answer the following questions and regain your sanity!

1. Is this valid code?:
   ```
   int &z = 5;
   ```
2. Is this a valid type?:
   ```
   int *&z;
   ```
3. Is this a valid type?:
   ```
   int&* x;
   ```
4. Can you create a reference to a reference? (&&!?)
5. Is this valid code?:
   ```
   int& arr[3] = {1, 2, 3};
   ```

1. No! An l-value reference can only be to something you can take the address of (l-value)

2. Yes! A reference to a pointer is valid

3. No! You can't have a pointer to a reference, when a reference might not even have a memory location

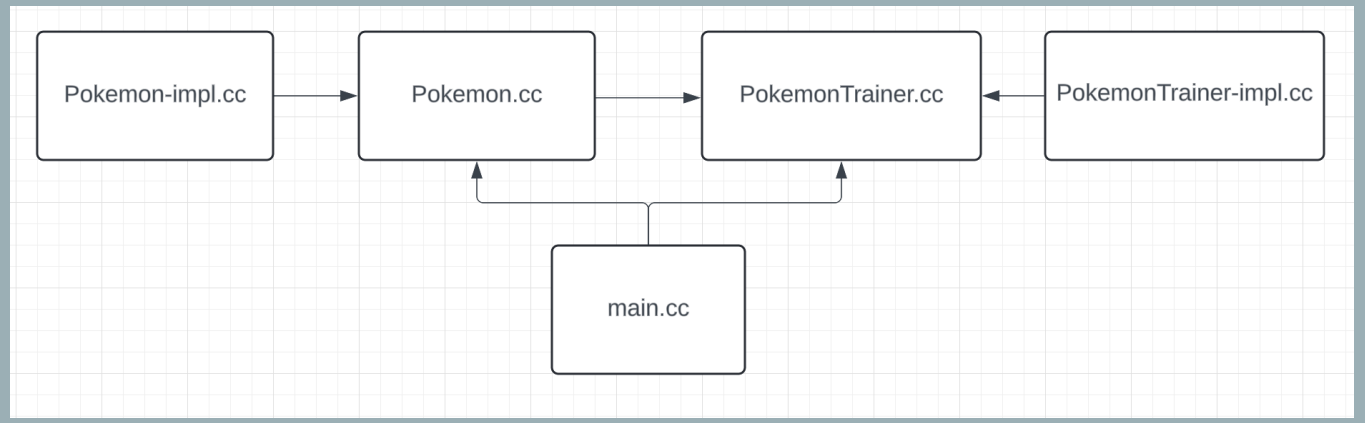
4. No! Because there is no real need to. (&& instead refers to a temporary reference, r-value)

5. No! You cannot create an array of references

3. Compilation (10 mins)

You are a young software developer working on a revolutionary, original and new game: Pokémon Purple. You've got basic combat simulation and basic Pokémon and Pokémon Trainer classes down, but something is not right; your code is not compiling! Pin down the compilation bugs in your code and fix the compilation commands below!

1. `g++20 Pokemon.cc -c`
2. `g++20 PokemonTrainer.cc -c`
3. `g++20 main.cc -c`



- Pokemon.cc: line 1, 6, and 12 should have "export" at the beginning
- Pokemon-impl.cc: line 1 should remove "export"
- PokemonTrainer.cc: should import Pokemon
- main.cc: line 2, 3 should remove <> around the user-defined modules

- ```
  g++20h iostream compare string
  g++20 Pokemon.cc -c
  g++20 Pokemon-impl.cc -c
  g++20 PokemonTrainer.cc -c
  g++20 PokemonTrainer-impl.cc –c
  g++20 main.cc -c
  g++20 *.o -o exc
  ```
  ```

4. MIL (5 mins)

Kindergartener Little Johnny, armed with C++ knowledge, is trying to write a gracefully elegant program about his favourite animal, the monkey. However, Johnny is young and inexperienced in software development and has made mistakes in his constructor. Help him perfect his code before he shows the teacher!

```cpp
// monkey.cc
import <iostream>;
class Monkey {
    bool happy = true;
    public:
        bool isHappy(int &bananas) {
            if (happy && bananas > 0) std::cout << "ook ook." << std::endl;
            else std::cout << "OOH OOH AAH AAH." << std::endl;
            return happy;
        }
        Monkey(bool h) : happy{h} {}
};
```

```cpp
class Johnny {
    Monkey monkey;
    const int bananas;
    int bananas2;
    int& bananas3;
    public:
        void speak() {
            if (monkey.isHappy(bananas3)) std::cout << "yay!" << std::endl;
            else std::cout << "no!" << std::endl;
        }
        Johnny() {
            monkey = monkey{true};
            bananas = 100;
            bananas2 = bananas;
            bananas3 = bananas2;
        }
};
```

```
class Johnny {
    Monkey monkey;
    const int bananas;
    int bananas2;
    int& bananas3;
    public:
        void speak() {
            if (monkey.isHappy(bananas3)) std::cout << "yay!" << std::endl;
            else std::cout << "no!" << std::endl;
        }
        Johnny() :
        monkey{true}, bananas{100}, bananas2{bananas}, bananas3{bananas2} {}
};
```

5. Operator Overloading (15 mins)

Professor Oak has tasked Ash Ketchum to help him implement the Pokémon class for the programming in his new and improved Pokédex, with basic combat simulation. Ash left school at the tender age of 11 to "catch 'em all", and so he has limited knowledge of OOP, let alone software development. Help Ash develop the spaceship (<=>) operator for the comparison of Pokémon level and type effectiveness against each other, as well as the input and output (>>, <<) operators for Pokédex entries.

Input format: c "index" "name" "level" "type"

eg. c 0 Bulbasaur 1 g

Output format: [Name: "name"] Level: "level" Type: "type"

eg. [Name: Squirtle] Level: 5 Type: WATER

```cpp
// *** No need to change the header file *** //
class Pokemon {
  public:
    inline static const unsigned int MAX_POKEMON = 6;
    enum class Type {FIRE, WATER, GRASS};
    std::string name;
    int level;
    Type type;
    auto operator<=>(const Pokemon&) const; // to implement
    std::string convertType(const Type& type) const;
};

std::ostream& operator<<(std::ostream&, const Pokemon&); // to implement
std::istream& operator>>(std::istream&, Pokemon&); // to implement

#endif
```

```cpp
istream& operator>>(istream& in, Pokemon& poke) {
    string n;
    in >> n;
    poke.name = n;

    int l;
    in >> l;
    if (in.good()) {
        if (l < 0 || l > 100) {
            cerr << "Invalid level!" << endl;
            return in;
        } else poke.level = l;
    } else {
        cerr << "Invalid level!" << endl;
        return in;
    }

    char t;
    in >> t;
    if (t == 'f') poke.type = Pokemon::Type::FIRE;
    else if (t == 'w') poke.type = Pokemon::Type::WATER;
    else if (t == 'g') poke.type = Pokemon::Type::GRASS;
    else {
        cerr << "Invalid type!" << endl;
        return in;
    }
    return in;
}
```

```cpp
ostream& operator<<(ostream& out, const Pokemon& poke) {
    out << "[Name: " << poke.name
        << "]  Level: " << poke.level
        << "  Type: " << poke.convertType(poke.type);
    return out;
}
```

```cpp
auto Pokemon::operator<=>(const Pokemon& opponent) const {
    auto result = level <=> opponent.level;
    if (result != 0) return result;
    if (type == opponent.type) return result;
    if (type == Type::FIRE) {
        if (opponent.type == Type::GRASS) result = strong_ordering::greater;
        else result = strong_ordering::less;
    } else if (type == Type::WATER) {
        if (opponent.type == Type::FIRE) result = strong_ordering::greater;
        else result = strong_ordering::less;
    } else {
        if (opponent.type == Type::WATER) result = strong_ordering::greater;
        else result = strong_ordering::less;
    }
    return result;
}
```

6. Iterator & Big 5 (30 mins)

You urgently need to access your email, but Duo 2-factor authentication appears on your screen. AI has been improving at an alarming rate, and as a result, Duo now requires you to complete the big 5 of the following class definition, as well as implement the Iterator pattern to login. Luckily, you are a student of CS 246, so you confidently attack the problem. Implement the big 5 and an Iterator.

```
class Duo {
    bool ***arr; // array of pointers to arrays of pointers
    int arrLen;
    int* lens; // array of lengths of inner array of pointers
    public:
        Duo() : arr{new bool **[1]}, arrLen{1}, lens{new int[1]} {
            arr[0] = new bool*[1];
            arr[0][0] = new bool{true};
            lens[0] = 1;
        }
        // write out the rest of big 5
        class Iterator {
            bool ***arr;
            int arrLen;
            int* lens;
            int arrPos; // position in outer array
            int inPos; // position in inner array
            Iterator(bool*** arr, int arrLen, int* lens, int arrPos, int inPos) : arr{arr},
             arrLen{arrLen},
             lens{lens},
             arrPos{arrPos} {}
            bool operator!=(const Iterator& o) {}
            Iterator& operator++() {}
            bool& operator*() {}
        };
        Iterator begin() {}
        Iterator end() {}
};
```

```
~Duo() {
    for (int i = 0; i < arrLen; ++i) {
        for (int j = 0; j < lens[i]; ++j) {
            delete arr[i][j];
        }
        delete[] arr[i];
    }
    delete[] arr;
    delete[] lens;
}
```

```
Duo(const Duo& other) : arr{new bool**[other.arrLen]}, arrLen{other.arrLen}, lens{new int[other.arrLen]} {
    for (int i = 0; i < arrLen; ++i) {
        lens[i] = other.lens[i];
        arr[i] = new bool*[lens[i]];
        for (int j = 0; j < lens[i]; ++j) {
            arr[i][j] = new bool{*(other.arr[i][j])};
        }
    }
}
```

```
Duo(Duo&& other) : arr{other.arr}, arrLen{other.arrLen}, lens{other.lens} {
    other.arr = nullptr;
    other.lens = nullptr;
other.arrLen = 0;
}
```

```cpp
Duo& operator=(const Duo& other) {
    if (this == &other) return *this;
    Duo tmp{other};
    std::swap(arr, tmp.arr);
    std::swap(arrLen, tmp.arrLen);
    std::swap(lens, tmp.lens);
    return *this;
}
```

```cpp
Duo& operator=(Duo&& other) {
    if (this == &other) return *this;
    std::swap(arr, other.arr);
    std::swap(arrLen, other.arrLen);
    std::swap(lens, other.lens);
    return *this;
}
```

```cpp
bool operator!=(const Iterator& o) {
    return (arr != o.arr) || (arrLen != o.arrLen) || (lens != o.lens) || (arrPos != o.arrPos) || (inPos != o.inPos);
}
```

```cpp
Iterator& operator++() {

    if (inPos == lens[arrPos] - 1) {
        ++arrPos;
        inPos = 0;
    }
    else {
        ++inPos;
    }
    return *this;
}
```

```cpp
bool& operator*() {
    return *(arr[arrPos][inPos]);
}
```

```cpp
Iterator begin() {
    return Iterator(arr, arrLen, lens, 0, 0);
}
Iterator end() {
    return Iterator(arr, arrLen, lens, arrLen, 0);
}
```

Other misc. things you should probably know

- ★ Steps of object creation (space is allocated, fields are initialized, constructor body runs)
- ★ Steps of object destruction (opposite to object creation order)
- ★ MIL initializes in declaration order, not the order they appear
- ★ Objects must always be initialized (cannot just declare an object)
- ★ As soon as you write your own constructor, the compiler-provided default constructor goes away
- ★ The destructor is automatically called on stack-allocated objects when they leave scope

QUESTIONS? CONCERNS? THOUGHTS?

Doubts? Inquiries? Queries? Interrogations? Examinations? Cross-Examinations? Quizzes? Disputes? Arguments? Debates? Rebuttals? Uncertainties? Controversies? Reservations? Dubieties? Issues? Problems? Matters? Points at issue? Business? Subjects? Topics? Themes? Items? Cases? Proposals? Propositions? Debriefs? Suspicions? Challenges? Probes? Catechisms? Ideas? Notions? Beliefs? Pre-Conceptions? Post-Conceptions? Convictions? Opinions? Views? Impressions? Images? Perceptions? Assumptions? Presumptions? Hypotheses? Theories? Suppositions? Postulations? Abstractions? Apprehensions? Conceptualizations? Intentions? Plans? Designs? Purposes? Interests? Aims? Reasonings? Contemplations? Musings? Ponderings? Considerations? Reflections? Speculations? Introspections? Deliberations? Studies? Ruminations? Cerebrations? Cogitations? Meditations? Broodings? Funny feelings? Hunches? Anticipations? Expectations? Prospects? Contemplations? Worries? Troubles? Bothers? Perturbations? Consternations? Disquietudes? Solicitudes? Anxieties? Fears? Hopes? Cares? Sympathies? Aspirations? Ambitions? Dreams? Reveries? Regards? Thanks?