Creative Software Programming Practice (week-14-1)

This exercise consists of several quizzes of `std::exceptions`. Guess the results before running.

You should be able to explain why you are getting such a result.

```
// p1.cc
#include <iostream>

static bool flag = true;

class ExceptionA: public std::exception {};
class ExceptionB: public std::exception {};

void f3() {
    if (flag) {
        throw "Exception";
    }
}

void f2() {
    f3();
    if (flag) {
        throw ExceptionA();
    }
}

void f1() {
    f2();
    if (flag) {
        throw ExceptionB();
    }
}

int main() {
    try {
        f1();
    } catch (ExceptionA& a) {
        std::cout << "exceptionA" << std::endl;

    } catch (ExceptionB& b) {
        std::cout << "exceptionB" << std::endl;

    } catch (...) {
        std::cout << "exception" << std::endl;
    }
    return 0;
}
```

```
// p2.cc
#include <iostream>
```

```cpp
static bool flag = true;

class ExceptionA: public std::exception {};
class ExceptionB: public std::exception {};

void f3() {
    if (flag) {
        throw "Exception";
    }
}

void f2() {
    try {
        f3();
    } catch (ExceptionA& a) {
        std::cout << "exceptionA" << std::endl;

    } catch (ExceptionB& b) {
        std::cout << "exceptionB" << std::endl;

    } catch (...) {
        std::cout << "exception" << std::endl;
    }
    if (flag) {
        throw ExceptionA();
    }
}

void f1() {
    try {
        f2();
    } catch (ExceptionA& a) {
        std::cout << "exceptionA" << std::endl;

    } catch (ExceptionB& b) {
        std::cout << "exceptionB" << std::endl;

    } catch (...) {
        std::cout << "exception" << std::endl;
    }
    if (flag) {
        throw ExceptionB();
    }
}

int main() {
    try {
        f1();
    } catch (ExceptionA& a) {
        std::cout << "exceptionA" << std::endl;

    } catch (ExceptionB& b) {
        std::cout << "exceptionB" << std::endl;

    } catch (...) {
        std::cout << "exception" << std::endl;
    }
    return 0;
}
```

```
// p3.cc
#include <exception>
#include <iostream>

class Parent : public std::exception {
 public:
  virtual const char* what() const noexcept { return "Parent!\n"; }
};

class Child : public Parent {
 public:
  const char* what() const noexcept { return "Child!\n"; }
};

int main() {
  try {
    throw Child();
  } catch (Parent& p) {
    std::cout << "Parent raised!" << std::endl;
    std::cout << p.what();
  } catch (Child& c) {
    std::cout << "Child raised!" << std::endl;
    std::cout << c.what();
  }
}
```

You can also explicitly indicate that a function does not raise an exception via the `noexcept` keyword. In this case, the exception is not actually thrown, it just ignored, so if an exception is raised in a function marked noexcept, the program may raise an error.

```
// p4.cc
#include <iostream>

int foo() noexcept {}

int bar() noexcept { throw 1; }

int main() {
  foo();
  try {
    bar();
  } catch (int x) {
    std::cout << "Error : " << x << std::endl;
  }
}
```

C++ has standard exception classes.

- std::bad_alloc
- std::range_error
- std::out_of_range

In addition, the default exception function allows you to check which exception is with the `what` method. And also you can override it.

```cpp
// p5.cc
#include <iostream>

class MemoryException: public std::exception {
public:
    virtual const char* what() const noexcept {
        return "memory exception";
    }
};

void f() {
    throw MemoryException();
}

int main() {
    try {
        f();
    } catch (MemoryException& e) {
        std::cout << e.what() << std::endl;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
    }
    return 0;
}
```