Week 5

CS 150 – C++ Programming I
In-Person Lecture

A Little Review

- We'll start with a little review of week 3 before we dive into today's exams, which will take the last 3 hours.
  - To review the material, I'm going to ask you questions
  - You are going to confer with your "group" (those in your row)
  - You'll answer using a "clicker" program
- Log into the desktop computers
- Double-click the file in Q:\faculty5\sgilbert\cs150\MWPM
- Log in with your Canvas ID (eg. sgilbert) and your student ID (eg. C01234567), just like the Homework Console

Streams & Files Review

- `<fstream>`: `ifstream`, `ofstream`, `fstream`
  - `ifstream in{"data.txt"}; // opens file`
  - `ofstream out{"out.txt"}; // creates file`
- Check if opened: `if (in.fail())...`
- May use *`fstream`* and "open mode"
  - `fstream log{"log.txt", ios::app};`
- Reading (extracting) data from a stream
  - `in.get(ch); getline(in, line);`
  - `in >> n; // fail on EOF or bad data`

Question

- If *`test.txt`* contains: `If I saw an Aardvark I would scream!` What is printed?
  - A. 5
  - B. 6
  - C. 0
  - D. 1
  - E. None of these

```
ifstream inFile("test.txt");
char ch;
int i = 0;
while (inFile.get(ch))
  if (tolower(ch) == 'a')
     i++;
cout << i << endl;
```

Question

- If *`file.txt`* contains:
  `Four and 20 blackbirds!`
  What is printed?

  - A. 20
  - B. Four and 20 blackbirds!
  - C. Four 20
  - D. 20 blackbirds!
  - E. None of these

```
ifstream inFile;
inFile.open("c:\\file.txt");
char ch; int n = 0;
while (inFile.get(ch)) {
  if (isdigit(ch)) {
      inFile.unget();
      inFile >> n;
      cout << n;
  }
}
```

Question

- If *test.txt* contains `Who Is 24601?` What is printed?
  - A. WHO IS 24601?
  - B. HOS?
  - C. WHOIS
  - D. WI
  - E. None of these

```
ifstream inFile("test.txt");
char ch;
while (inFile.get(ch))
  if (isupper(ch))
    cout << toupper(ch);
```

Question

```
string name; int scores;
ifstream inFile("results.txt");
```

- *results.txt* contains lines of text like this:

```
Smith 94
Jones 75
```

- What is the legal way of reading a student's name and the student's scores in the "results.txt" file?
  - A. `inFile >> name >> scores;`
  - B. `inFile << name << scores;`
  - C. `getline(inFile, name); inFile >> scores;`
  - D. `getline(inFile, name);`
    `getline(inFile, scores);`

String Stream Review

- `<sstream>`: `istringstream, ostringstream`
- Output string streams for formatted output
  - `ostringstream out;   // empty`
  - `out.put('Q'); out << "ED. " << 2018 << endl;`
  - `string s = out.str();   // when finished`
  - `out.str("");  // fresh string buffer`
- Input string streams (reading & parsing)
  - `istringstream in("Jan 1, 2018");`

  -
    ```
    string month; in >> month;
    int day, year; in >> day;
    in.get();  in >> year;
    ```

Question 1

- What goes in the blank line?
  - A. istringstream
  - B. ostringstream
  - C. istreamstring
  - D. stringstream;
  - E. istream

```
double mystery(const string& s)
{
    ____________________x(s);
    double n;
    x >> n;
    return n;
}
```

Question 2

- What goes in the blank line?

  - A. osstream
  - B. ostreamstring
  - C. ostringstream
  - D. istringstream;
  - E. ofstream

```
string mystery(double n)
{
    ________________x;
    x << n;
    return x.str();
}
```

Question 3

```
istringstream in;
in.str(" 3.1459  ");
double n = 3;
in >> n;
```

- After running, `in.fail()` is ________, `in.eof()` is __________, and n is _______________?

  - A. `false, false, 3`
  - B. `false, true, 3.1459`
  - C. `false, true, 3`
  - D. `false, false, 3.1459`
  - E. `true, true, 3.1459`

Assumptions & Assertions Review

- Assumptions about valid inputs and outputs
  - Preconditions are inputs, postconditions are outputs
  - `@pre n should be >= 0 // sqrt precondition`
  - `@post status true if number read correctly`
- What to do about precondition violations?
  - Fix it silently, terminate with message, return error code, throw an exception, ignore it
- Use *`assert`* to automatically detect programming errors

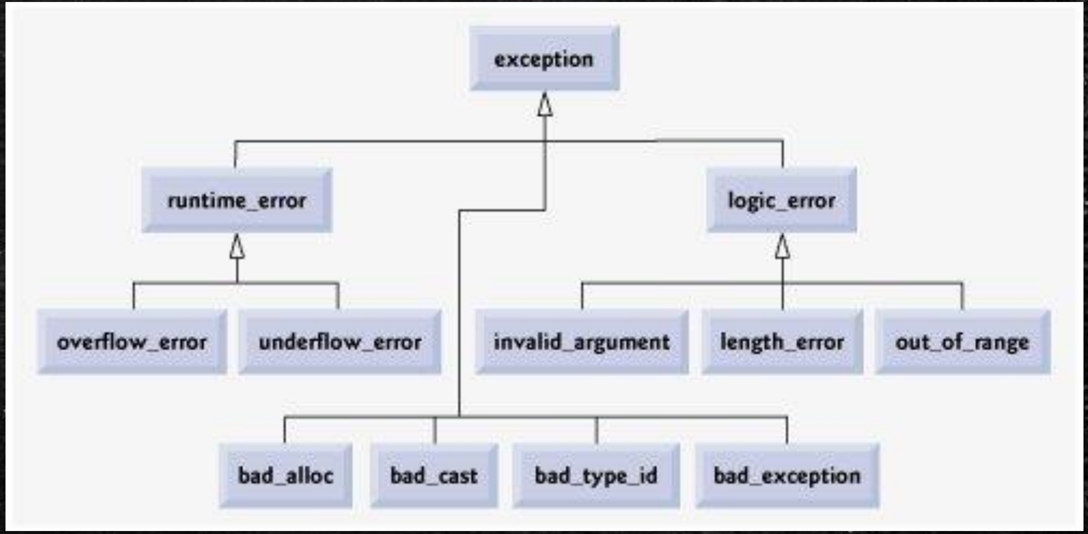
  -
    ```
    int sum_between(int lower, int upper) {
        assert(lower <= upper); // cannot happen
    ```

Exceptional Situations

- Use exceptions when you can detect an error, but don't know how it should be handled in every situation
  - Especially important for library functions
  - Print message? Write to a log? Terminate? You CAN'T tell
- Exception handling fundamentals
  - Use `throw` to signal when an error occurs
    - `if (arg > 255) throw illegal_argument("message");`
  - Wrap function call (which may fail) in a `try` block
  - Follow with `catch` or `catch(. . .)`

Standard Exceptions

- In C+ you can throw any kind of object
  - `if (x > 0) throw 15; // int error code; can't ignore`
  - You may create UDT to signify a custom type of error
- The standard library exceptions similar to Java's
  - `#include <stdexcept>`
  - Class hierarchy: catch most specific to most general
  - Catch exception by reference
  - Call member function `what()`



Question

- What prints?
  - A. a
  - B. b
  - C. c
  - D. abc
  - E. Cannot tell from this code

```
string s("hello");
try {
    s.at(s.size()) = 'x';
    cout << "a";
}
catch (out_of_range& e)  { cout << "b"; }
catch (...) { cout << "c"; }
```

Question

- What prints?
  - A. a
  - B. b
  - C. c
  - D. abc
  - E. Cannot tell from this code

```
string s("hello");
try {
    if (s.size() > 2) throw s.size();
    s.at(s.size()) = 'x';
    cout << "a";
}
catch (out_of_range& e)  { cout << "b"; }
catch (...) { cout << "c"; }
```

Question

- What prints?
  - A. a
  - B. b
  - C. c
  - D. abc
  - E. This is undefined.

```
string s("hello");
try {
    s[1000] = 'x';
    cout << "a";
}
catch (out_of_range& e)  { cout << "b"; }
catch (...) { cout << "c"; }
```

Template Review

- A recipe or blueprint for writing a function or class

```
void swap(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
```

```
int x = 3, y = 4;
swap(x, y); // 4, 3
```

```
double n = 3.5, m = 1.5;
swap(n, m);  // Error
```

```
template <typename  T>
void swap(T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
}
```

```
double n = 3.5, m = 1.5;
swap(n, m);  // 1.5,  3.5
```

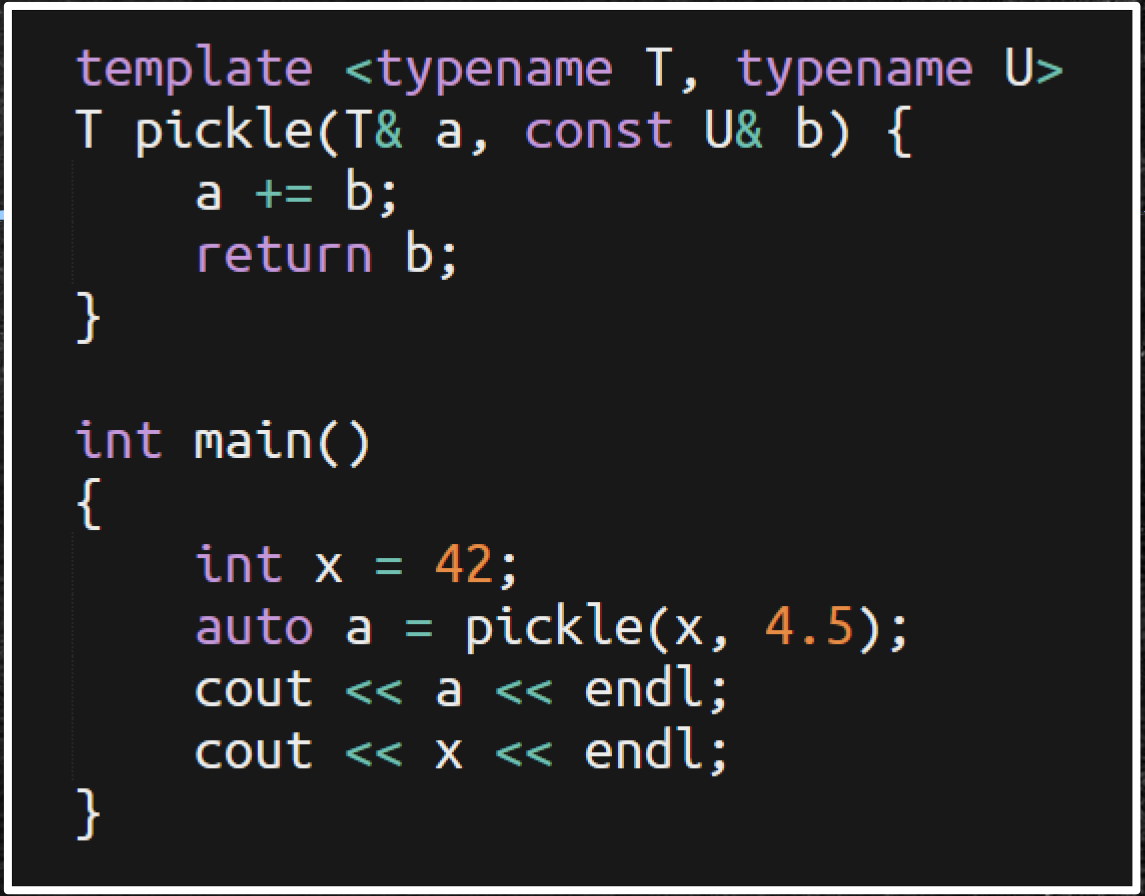
Some Template Details

- Function templates are not pre-compiled
  - Usually placed in header files (no `using namespace std`)
- Function template generates multiple template functions

  -
    ```
    template <class T> // may use class instead
    T add(T a, T b) { return a + b; }
    ```
  - Explicit instantiation: `add<int>(3.5, 2.5);`
  - Implicit instantiation: `add(2.5, 1.5);`
- May have multiple type parameters
  - `template <class T, class U>` ...

Question

- What prints in `main()`?
  - A. `4.5, 46.5`
  - B. `4, 46.5`
  - C. `4.5, 46`
  - D. `4, 46`
  - E. None of these

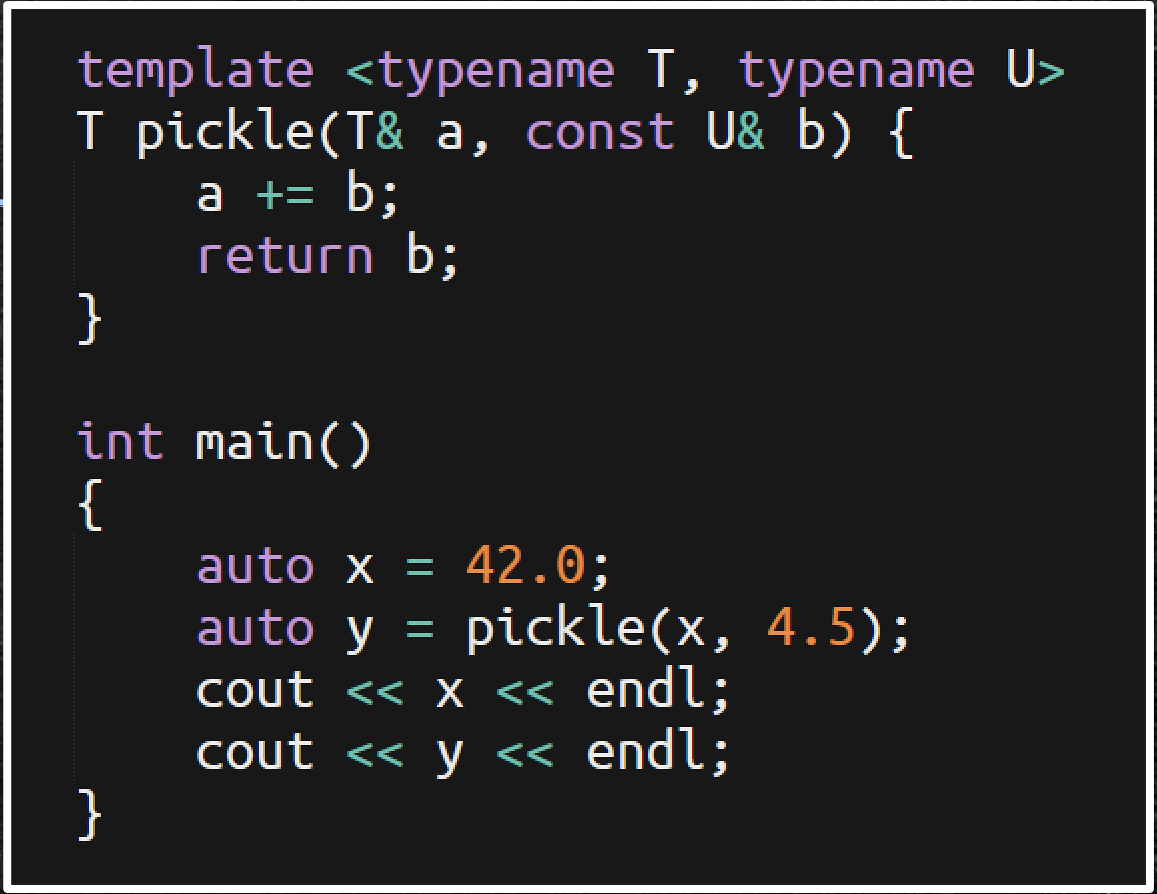


```
template <typename T, typename U>
T pickle(T& a, const U& b) {
    a += b;
    return b;
}

int main()
{
    int x = 42;
    auto a = pickle(x, 4.5);
    cout << a << endl;
    cout << x << endl;
}
```

Question

- What prints in `main()`?
  - A. `46.5, 4.5`
  - B. `46.5, 4`
  - C. `46, 4.5`
  - D. `46, 4`
  - E. None of these



```
template <typename T, typename U>
T pickle(T& a, const U& b) {
    a += b;
    return b;
}

int main()
{
    auto x = 42.0;
    auto y = pickle(x, 4.5);
    cout << x << endl;
    cout << y << endl;
}
```

Question

```cpp
template <typename T>
void mystery(T a, T b) {
    cout << a << " + " << b << "->"
        << (a + b) << endl;
}
```

- Which of these is illegal?
  - A. `mystery(3L, 4L);`
  - B. `mystery(3.F, 4.F)`
  - C. `mystery("bob", "ray");`
  - D. All of these
  - E. None of these

Review: Structures & Enumerated Types

- Structure definition is a blueprint (usually in header file)
  - `struct Point { int x, y; };`
  - `struct Point {int x = 0, y = 0;} // c++11`
- Instantiation & initialization:

  -
    ```
    Point a;                // uninitialized
    Point b{};              // default init C++11
    Point c = {20, 50};     // aggregate all versions
    ```
- Enumerated Types: user-defined scalar (integral) type

  -
    ```
    enum class Coin {
        penny = 1, nickel = 5, dime = 10, quarter = 25
    };
    ```

Review: Structure Operations

- Member access operator is the "dot"
  - `a.x = 3;             // write to a structure member`
  - `cout << b.y;         // read from a member`
  - `s.location.x;        // nested Star structure`
- Only built-in aggregate operation is assignment
  - `if (a == b) // illegal (won't compile)`
- Using enumerated types
  - `Coin x = Coin::penny;`
  - `int value = static_cast<int>(x);`
  - `switch (x) {. . .}`

Question

- What prints?
  - A. 0
  - B. Undefined random value
  - C. Compiler error because x is private
  - D. Syntax error, mistake in structure definition

```
struct Val {
    double x;
}
. . .
Val v;
cout << v.x << endl;
```

Question

- What prints?
  - A. 0
  - B. Undefined random value
  - C. Compiler error because x is private
  - D. Syntax error, mistake in structure definition

```
struct Val {
    double x;
};
. . .
Val v;
cout << v.x << endl;
```

Question

- What prints out?
  - A. 0
  - B. 2.5
  - C. 2.5 but only in C++ 11 or 14
  - D. Compiler error because x is private
  - E. Syntax error, mistake in syntax

```
struct Val {
    double x;
};
. . .
Val v = {2.5};
cout << v.x << endl;
```

Ques

```
enum class Coin {
    penny = 1, nickel = 5, dime = 10,
    quarter = 25, half = 50
};

int valueOf(Coin c) { ______________;}
```

- What goes on the blank line?
  - A. `return c;`
  - B. `return c.value()`
  - C. `return static_cast<int>(c);`
  - D. Compiler error; wrong syntax for Coin
  - E. Need a `switch`; can't do it in one statement

Ques

```
enum class Coin {
    penny = 1, nickel = 5, dime = 10,
    quarter = 25, half = 50
};
Coin toCoin(int n) {
    switch (n) {
        case 1: ______________;
```

- What goes on the blank line?
  - A. `return Coin{n};`
  - B. `return penny;`
  - C. `return Coin.penny;`
  - D. `return Coin::penny;`

Review: vector Basics

- Create variables by specifying the base type
  ```
  – vector<int> v1,     // empty
        v2(10),         // 10 ints 0 initialized
        v3(3, 4),       // 3 ints initialized to 4
        v4{3, 4};       // 2 ints - 3 and 4
  ```
- Access with `at()`, `front()`, `back()`, or `[]`
- Members: `push_back()`, `pop_back()`, `size()`
- Loops: `for(auto e : v)... // &, etc`

Question

```
vector<int> v(3, 2);
cout << v.front() << v.size() << endl;
```

- What prints?
  - A. 32
  - B. 23
  - C. 33
  - D. 22
  - E. None of these

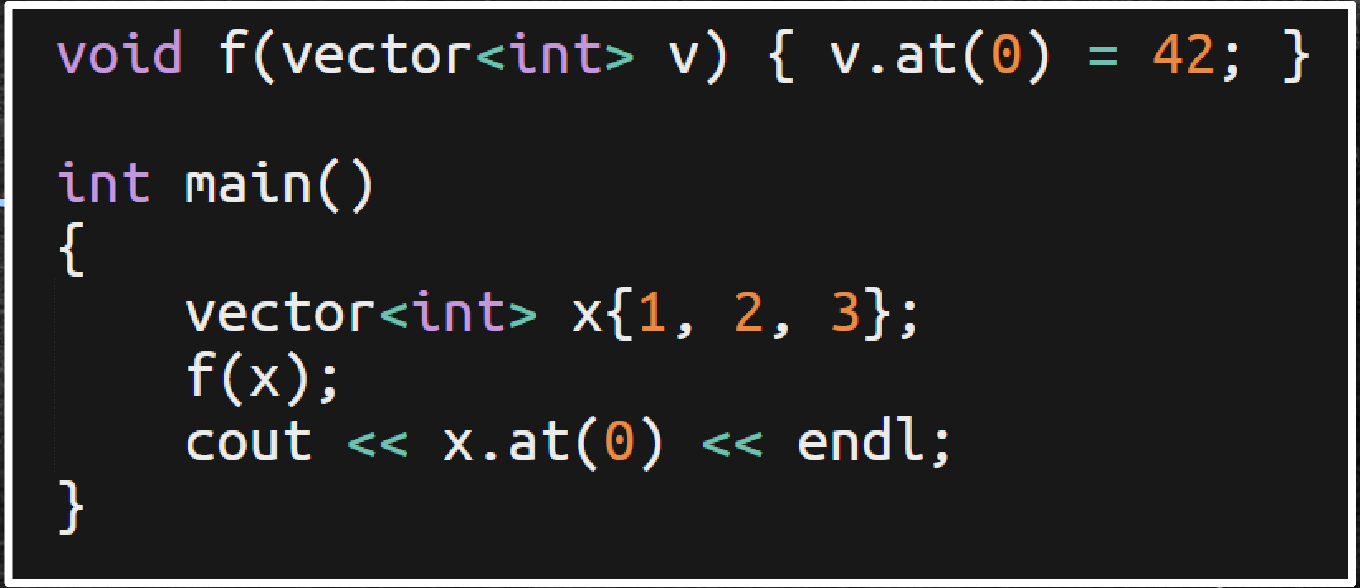
Question

```
vector<int> v{3, 2};
cout << v.front() << v.size() << endl;
```

- What prints?
  - A. 32
  - B. 23
  - C. 33
  - D. 22
  - E. None of these

Question

- What prints?
  - A. Compile time error
  - B. Runtime error
  - C. 42
  - D. 1
  - E. Unknown value

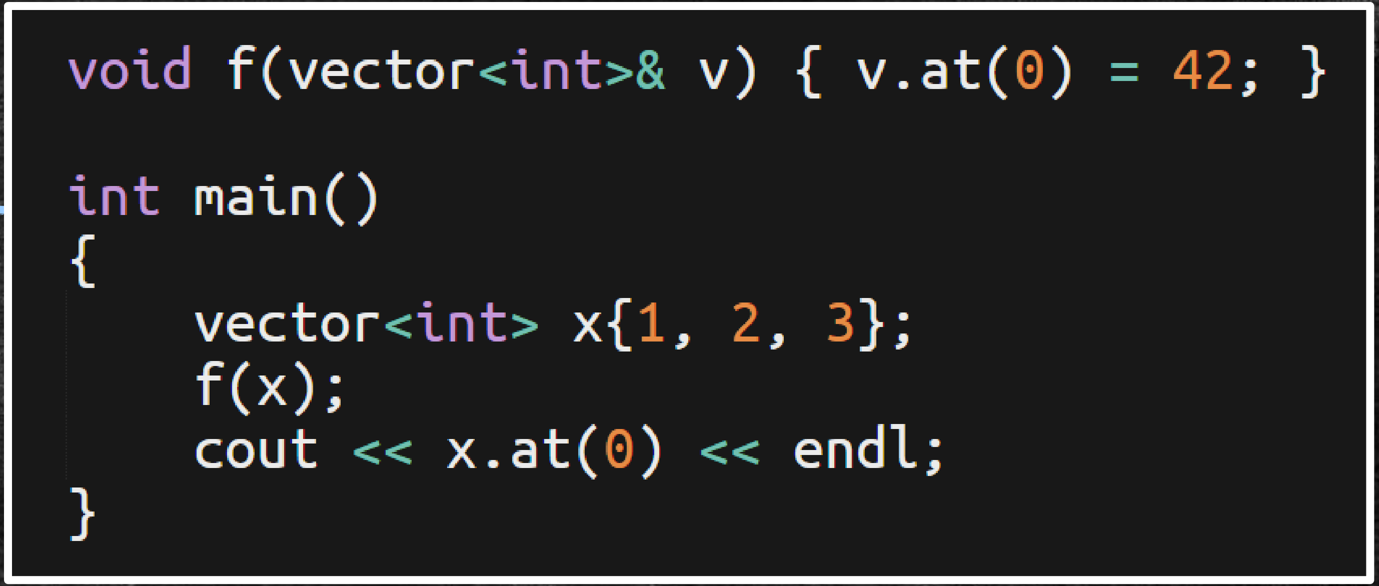


```cpp
void f(vector<int> v) { v.at(0) = 42; }

int main()
{
    vector<int> x{1, 2, 3};
    f(x);
    cout << x.at(0) << endl;
}
```

Question

- What prints?
  - A. Compile time error
  - B. Runtime error
  - C. 42
  - D. 1
  - E. Linker error

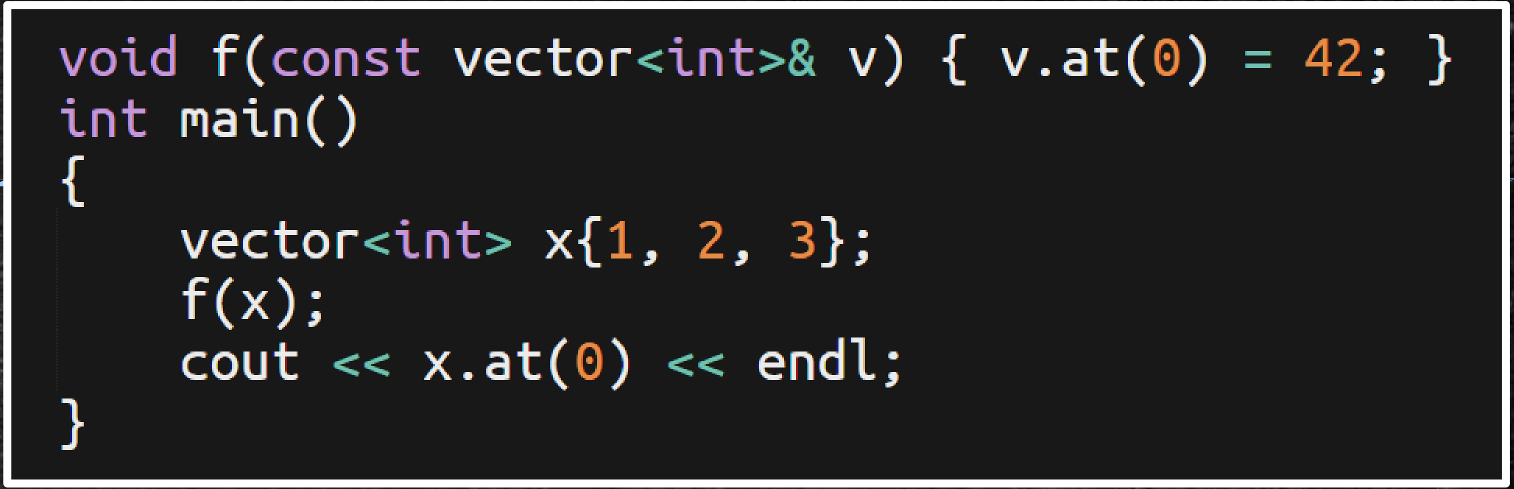


```
void f(vector<int>& v) { v.at(0) = 42; }

int main()
{
    vector<int> x{1, 2, 3};
    f(x);
    cout << x.at(0) << endl;
}
```

Question

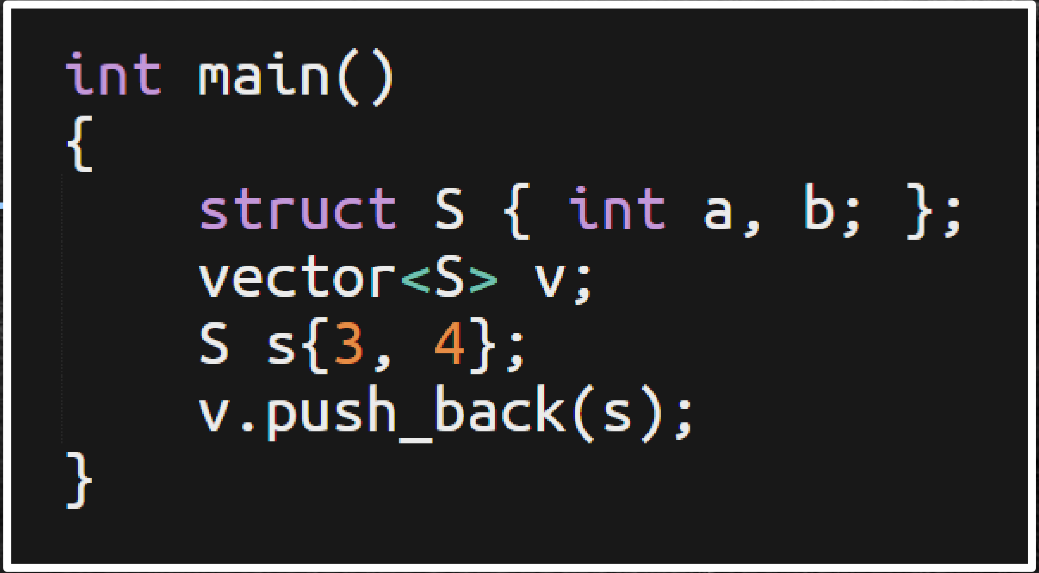
- What prints?
  - A. Compile time error
  - B. Runtime error
  - C. 42
  - D. 1
  - E. Linker error



```cpp
void f(const vector<int>& v) { v.at(0) = 42; }
int main()
{
    vector<int> x{1, 2, 3};
    f(x);
    cout << x.at(0) << endl;
}
```

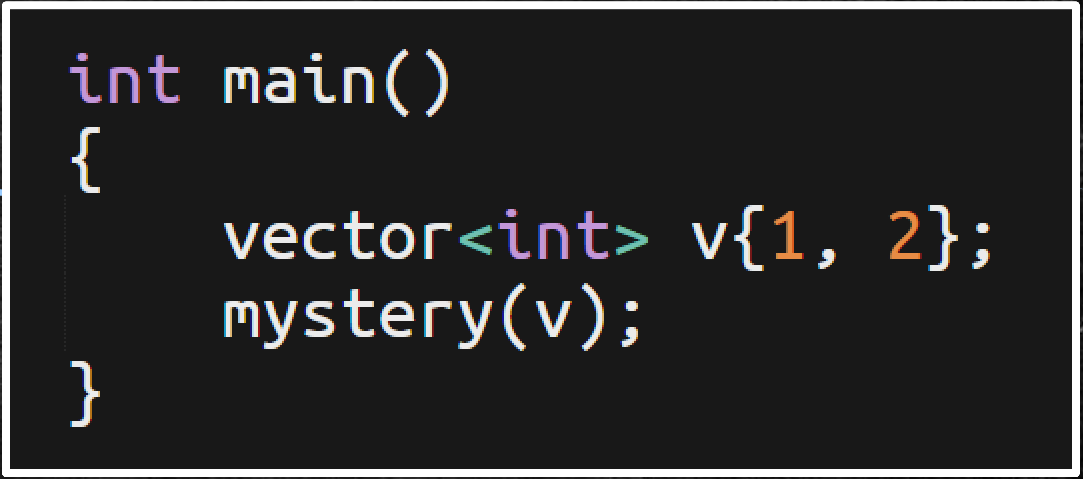
Question

- What line of code will print the value 4?
  - A. `cout << v.b << endl;`
  - B. `cout << v.b.at(0) << endl;`
  - C. `cout << v.at(0).b << endl;`
  - D. `cout << v.at(0)[1] << endl;`
  - E. None of these



```
int main()
{
    struct S { int a, b; };
    vector<S> v;
    S s{3, 4};
    v.push_back(s);
}
```

Question



```
int main()
{
    vector<int> v{1, 2};
    mystery(v);
}
```

- What is the correct prototype for *`mystery()`*?
  - A. `void mystery(vector<int>);`
  - B. `void mystery(vector<int>&);`
  - C. `void mystery(vector&);`
  - D. `void mystery(const vector<int>&);`
  - E. Either B or D could be correct.

LEC-5A Preview-Memory and Pointers

- Different storage areas in memory (stack, heap, static)
  - Naming concepts: scope, storage (duration), linkage
  - How to create and use global and local variables
- Variable characteristics: name, type and value
  - Variable storage and the `sizeof` operator
  - Variable location and the address operator
- Pointers-variables which contain addresses
  - Defining and initializing pointer variables
  - Dereferencing and assigning to a pointer
  - Calling by pointer, null and const pointers

LEC-5B Preview-Graphics & Digital Filters

- Digital images and RGB colors
- Processing image files using the stb image libraries in C++
  - Loading an image (onto the heap) with `stbi_load()`
  - Using pointer output parameters and C-style strings
  - Saving an image in different formats
- Writing digital filters to modify an image
  - Using address arithmetic and iterator loops
  - Process filters: the `darken` and `blue` filters
  - State filters: the `vertical-stripes` filter
  - Structures & `reinterpret_cast`: `horizontal-stripes`

LEC-5C Preview-Introducing Arrays

- The array type and how it differs from `vector`
  - Defining and initializing arrays
  - The allocated size of an array
  - Selecting elements from an array using the subscript operator
  - Array characteristics: copy and compare
- Arrays and pointers: similarities and differences
  - Address arithmetic and dereferencing with arrays
  - Passing arrays to functions and pointer decay
  - Using `const` with arrays
  - Processing arrays with loops

LEC-5A Preview-Arrays & Algorithms

- A common C++ idiom: `*beg++`
- Common array algorithms
  - Counting & cumulative algorithms
  - Extreme values: returning pointers or indexes
  - Adding separators with the fencepost algorithm
- Searching algorithms and their efficiency
  - Linear search: the `find()` function
  - Big-O notation and `O(n)` functions
  - Binary search: the `bfind()` function `O(log n)`

Week 5 Homework Preview

- Week 5 HW due by 1pm July 17th (Mon) or 18th (Tue)
  - H22 - Pointers & Graphics: `negative()`
  - H23 - Pointers & Graphics: `greenScreen()` & `composite()`
  - H24 - Images & Structures: `flip()` & `mirror()`
  - H25 - Arrays: `alternatingSum()`, `minMax()` & `sameSet()`
  - H26 - More Array Functions: `copyEvens()`, `cliqueCount()` & `sevenEleven()`

Programming Exam 4, 5 and Midterm #2

- Now – Programming Exam #4
  - I will collect your cellphones, watches & electronics
  - Place all books, backpacks, notes at front or back of the room
  - Move to your assigned seat; do not log in
  - I will start PE04 on your computer
  - Log in using your Homework Console credentials
  - When you are done, submit the exam and leave
- Come back by 3pm when PE 05 will start
- Come back by 4pm when Midterm Exam #2 will start