CS 246 Midterm Review Session

Spring 2023

Question 1: What are the 3 I/O streams and 2 I/O operators?
Answer: `cin, cout, cerr` and `<<, >>`

Question 2: Write a program that divides 2 numbers. When dividing by 0, output undefined.
Answer:

```
int main() {
  float x, y;
  cin >> x >> y;
  if (y == 0) {
    cout << "undefined" << endl;
  }
  cout << x/y << endl;
}
```

Question 3: Write a program that reads all integers from standard input and outputs the sum. Exit gracefully, stop on bad inputs or EOF.
Answer:

```
int main() {
  int sum, i;
  sum = 0;
  while (cin >> i) {
    sum += i;
  }
  cout << sum << endl;
}
```

Question 4: Fill in the blanks: Pointers are like constant pointers with automatic dereference.

Question 5: True or False

(a) You can leave a lvalue reference uninitialized: `int &x;`

(b) You can create a pointer to a reference: `int &*x;`

(c) You can create a reference to a pointer: `int *&x;`

(d) You can create a reference to a reference: `int &&x`

(e) You can create an array of references: `int &r[3] = {...}`

(f) You can pass a reference as a function parameter

Answer: False; False; True; False; False; True

Question 6: In `istream& operator>>(istream &in, int &n)`, why is the stream being taken and returned as a reference.
Answer: We want to chain `cin`; We don't want to create a copy of `istream`(not allowed).

Question 7: Given

```
struct Node {...}
Node *np = new Node;
...
delete np;
```

Is the pointer on the heap or stack? What about the Node? **Answer:** Stack; Heap.

Question 8: Fix the bug

```
Node *nodeArray = new Node[10];
...
delete nodeArray;
```

Answer: You need `delete [] nodeArray`

Question 9: Given

```
struct Vec {
  int x, y, z
}
```

Write the body of

(a) `Vec operator+(const Vec &v1, const Vec &v2);`

(b) `Vec operator*(const int k, const Vec &v);`

(c) `Vec operator*(const Vec &v, const int k);`

(d) `ostream &operator<<(ostream &out, const Vec &v);`

(e) `istream &operator>>(istream &in, Vec &v);`

Answer:

(a)
```
Vec operator+(const Vec &v1, const Vec &v2) {
  return {v1.x+v2.x, v1.y+v2.y, v1.z+v2.z};
}
```

(b)
```
Vec operator*(const int k, const Vec &v) {
  return {k*v.x, k*v.y, k*v.z};
}
```

(c)
```
Vec operator*(const Vec &v, const int k) {
  return k*v;
}
```

(d)
```
ostream &operator<<(ostream &out, const Vec &v) {
  out << v.x << v.y << v.z;
  return out;
}
```

(e)
```
istream &operator>>(istream &in, Vec &v) {
  in >> v.x >> v.y >> v.z;
  return in;
}
```

Question 10: Given

```
struct Node {
  int data;
  Node *next;
}
```

(a) Why is the built-in copy constructor incorrect?

```
Node *n = new Node{1, new Node{2, new Node{3, nullptr}}};
Node m = *n;
```

(b) Write a copy constructor for `Node`

(c) Write a copy assignment operator for `Node`

(d) Write a move constructor for `Node`

(e) Write a move assignment operator for `Node`

Answer:

(a) Shallow copy.

(b)
```
Node(const Node &other): data{data}, next{other.next ?
                                 new Node{*other.next} : nullptr} {}
```

(c)
```
Node &operator=(const Node &other) {
  if (this == &other) return *this;
  data = other.data;
  delete next;
  next = other.next ? new Node{*other.next} : nullptr;
  return *this;
}
```

(d)
```
Node (Node &&other): data{other.data}, next{other.next} {
  other.next = nullptr;
}
```

(e)
```
Node &operator=(Node &&other) {
  std::swap(data, other.data);
  std::swap(next, other.next);
  return *this;
}
```

Question 11:

```
class list {
   struct Node;
   Node *theList;
 Public:
   class iterator {
     explicit Iterator(Node *p): p{p} {}
     bool operator!=(const Iterator &other) const {
       return p != other.p;
     }
     Iterator &operator++() {
       p = p->next;
       return *this;
     }
     int &operator*() {
       return p->data;
     }
   }
   Iterator begin() {
     return Iterator{theList};
   }
   Iterator end() {
     return Iterator{nullptr};
   }
}
```

(a) Fill in the blank(already filled)

(b) Write a traditional for loop that prints every other list item, starting from the front
Answer:

```
bool flag = true;
for ( auto it = lst.begin(); it != lst.end(); ++it ) {
    if ( flag ) cout << *it << ' ';
    flag = ! flag;
}

// OR

for ( auto it = lst.begin(); it != lst.end(); ++it ) {
  cout << *it << ' ';
  ++it;
}
```

(c) Write a range-based for loop that prints all positive items that exist list items to.

```
for (auto n: l) {
  if (n > 0) cout << n << ' ';
}
```