CS100 Computer Programming

Quiz 1

April 19, 2023

Answer the questions according to the C++17 standard.

For the compiler-generated special member functions, ignore whether they are `constexpr`, and ignore whether they are `noexcept` except for move operations.

1. (15 points) Your name: _____. Your student ID: _____.

   Your email: _____`@shanghaitech.edu.cn`

2. (10 points) Select the pieces of code that have (or may lead to) undefined behaviors.

   A.
   ```cpp
   std::vector<double> vec;
   for (std::size_t i = 0; i != n; ++i)
     std::cin >> vec[i];
   ```

   B.
   ```cpp
   void extend(std::vector<double> &vec) {
     for (auto x : vec)
       vec.push_back(x);
   }
   ```

   C.
   ```cpp
   std::size_t npos = -1;
   ```

   D.
   ```cpp
   int main() {
     std::string str;
     std::cout << str << std::endl;
   }
   ```

   E.
   ```cpp
   int *ptr = nullptr;
   delete ptr;
   ```

Solution:

A. `vec` is an empty vector. The behavior of the subscript operation `vec[i]` is undefined. This code has undefined behaviors unless `n == 0`.

B. `vec.push_back(x)` may cause reallocation of the storage, thus invalidating all iterators. The range-based for loop is based on iterators. Undefined behavior unless `vec` is empty.

C. `std::size_t` is unsigned and never overflows.

D. `std::string str;` default-initializes `str` to be an empty string. There is no undefined behavior here.

E. Passing a null pointer to `delete` does not do anything. In detail:

- No destructor is called.
- The deallocation function may or may not be called (it's unspecified), but the deallocation function is guaranteed to do nothing.

3. (10 points) Let `ival` be an `int`, and let `ptr` be of some pointer type. Select the expressions that yield an **rvalue**.

   A. `++ival` B. `ival++` C. `*&ptr` D. `&*ptr` E. `ptr[ival]` F. `*(ptr + ival)`

> **Solution:** `++ival` returns reference to `ival`, which is an lvalue. `ival++` returns the original value, which is an rvalue.
>
> `*&ptr` returns reference to `ptr`, lvalue. `&*ptr` returns the address of the object that `ptr` points to, which is the value that `ptr` holds, but not `ptr` itself. This is an rvalue.
>
> `ptr[ival]` and `*(ptr + ival)` are equivalent, both returning reference to `ptr[ival]`, lvalue.

4. (10 points) Let class `X` be defined as follows.

```
struct X {
  int a, b;
  std::string s;
  X() = default;
  X(X &&) = default;
  ~X() { std::cout << "Goodbye world" << std::endl; }
};
```

   Which of the following statements are true?

   A. In the destructor `~X`, the destructor of `std::string` is called to destroy the member `s` before `"Goodbye world"` is printed.

   B. The compiler will generate a default constructor for `X` (if it is used) which default-initializes all the members.

   C. The compiler will generate a move constructor for `X` (if it is used) as if it were defined as

```
X(X &&other) noexcept
    : a(std::move(other.a)), b(std::move(other.b)), s(std::move(other.s)) {}
```

   D. The compiler will generate a move constructor for `X` (if it is used) as if it were defined as

```
X(X &&other) noexcept
    : a(other.a), b(other.b), s(other.s) {}
```

   There is no need to apply `std::move` to the members of `other`, because `other` is an rvalue.

> **Solution:**
>
> A. `s` is destroyed **after** the execution of the function body.
>
> B. Correct.
>
> C. Correct.
>
> D. An rvalue reference is an lvalue. `std::move` is necessary, especially for `other.s`.

5. (10 points) Suppose `Dynarray` has both a copy assignment operator and a move assignment operator. Select the situation(s) where the **copy assignment operator** of `Dynarray` is used.

A.
```
Dynarray concat(const Dynarray &a, const Dynarray &b) {
  Dynarray result(a.size() + b.size());
  // Concatenates the contents of `a` and `b`. Details are omitted.
  return result;
}
int main() {
  Dynarray a, b;
  a = concat(a, b);
}
```

B. `Dynarray a; Dynarray b = a;`

C. `Dynarray *a, *b; a = b;`

D. `std::vector<Dynarray> vec(10); Dynarray a; vec[0] = a;`

Solution:

A. `concat(a, b)` is an rvalue. The assignment is a move assignment.

B. `Dynarray b = a;` uses the copy constructor.

C. `a = b` is an assignment of pointers, which does not involve any member function call.

D. `vec[0] = a;` is a copy assignment.

6. (10 points) Which of the following statements regarding **`const` member functions** is/are true?

   A. `const` member functions can only be called on `const` objects.

   B. `const` member functions cannot call non-`const` member functions (without a `const_cast`).

   C. In a `const` member function of class `X`, the implicit `this` pointer has type `const X *`.

   D. If a non-`const` member function does not modify any data member, the compiler will make it a `const` member function.

7. (10 points) Select the situation(s) where the **default constructor** of the class `X` is used.

   A. `X a[100];` B. `auto p = new X;` C. `X a();` D. `void fun(X a) {}`

8. (15 points) For each piece of code, write down the type of `var`.

   (a)
```
auto ival = 42; auto *var = &ival;
```

   (a) `int *`

   (b)
```
std::vector v(10, 3.14);
auto var = v[0];
```

   (b) `double`

   (c)
```
std::vector<std::vector<std::string>> vvs;
for (const auto &vs : vvs)
  for (const auto &var : vs)
    do_something(var);
```

   (c) `const std::string &`