Solutions, C++ Programming examination

2023-03-17

1. a) `example1` has UB, as `make_person` returns a reference to a temporary value

b) `example2` has UB, as `make_student` returns a reference to a local variable

c) `example3` is correct, as `make_teacher` returns a pointer to a dynamically allocated object

d) `example4` is correct, as it uses a local variable `t`

2. An array is not a pointer: it is an object containing all of its elements, and `sizeof` gives the size of a type.

`arr` is an array of 5 ints, and `sizeof(arr)` gives the size of the *array object* which is 20 bytes (i.e., on this machine, `sizeof(int) == 4`.

`&arr` is a pointer to the array (i.e., an `int*`), and on this machine a pointer is 8 bytes.

Regarding "an array is just a pointer to the first element": When using an array variable, it *decays* to a pointer. That is why the first two lines are equivalent: the expression `arr` gives a pointer to the first element.

3. The expression `pos =! last` is probably a typo, and its meaning is clearer if it is formatted `pos = !last`. As `last` is an int, it is implicitly converted to `bool`, so the value of `!last` is `true` or `false`.

Then, in the assignment `pos = !last` it is converted back to int as `0` or `1`, and finally that is interpreted as a boolean value. As `last` gets passed the argument 5, `!last` is `false` and the for loop is not entered.

4. The class `Time` needs `operator<<`, `operator>>`, and `operator+`, a default constructor, and a suitable constructor and/or accessors. It would also be a good idea to check that the constructor arguments are valid, but that is not required by the program in the problem (provided that the error checking is done in `operator>>()`).

```
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>

class Time {
  public:
    Time(int hh, int mm) : h(hh), m(mm) {}
    Time() = default;
    friend std::istream& operator>>(std::istream&, Time&);
    int get_h() const { return h; }
    int get_m() const { return m; }

  private:
    int h{};
    int m{};
};

std::istream& fail(std::istream& is)
{
    is.setstate(std::ios_base::failbit);
    return is;
}
```

```
std::istream& operator>>(std::istream& is, Time& t)
{
    int h;
    if (!(is >> h)) {
        return is;
    } else if (h < 0 || h > 23) {
        return fail(is);
    }

    char c;
    if (!is.get(c)) {
        return is;
    } else if (c != ':') {
        return fail(is);
    }

    int m;
    if (!(is >> m)) {
        return is;
    } else if (m < 0 || m > 59) {
        return fail(is);
    }

    t.h = h;
    t.m = m;
    return is;
}
std::ostream& operator<<(std::ostream& os, const Time& t)
{
    return os << t.h << ':' << std::setfill('0') << std::setw(2) << t.m;
}

Time operator+(const Time& a, const Time& b)
{
    int h = a.get_h() + b.get_h();
    if (h > 23) {
        h -= 24;
    }
    int m = a.get_m() + b.get_m();
    if (m > 59) {
        h += 1;
        m -= 60;
    }
    return Time(h, m);
}
```

5. A possible solution is:

```
#include <algorithm>
template <typename Iter, typename T>
class result_iter {
  public:
    result_iter(Iter first, Iter last, const T& t) : f(first), l(last), val(t)
    {
        next();
    }

    result_iter& operator++()
    {
        ++f;
        next();
        return *this;
    }
    T& operator*() { return *f; }
    bool operator!=(Iter it) const { return f != it; }

  private:
    void next() { f = std::find(f, l, val); }

    Iter f;
    Iter l;
    T val;
};

template <typename Iter, typename T>
result_iter<Iter, T> find_all(Iter first, Iter last, const T& val)
{
    return result_iter<Iter, T>(first, last, val);
}
```

6. `x` is captured by reference, and `operator()(int)` should compare its argument to the captured variable.

```
class my_less_than {
public:
    my_less_than(const int& r) : x(r) {}
    bool operator()(int val) { return val < x; }
private:
    const int& x;
};
```