DS 4^{th} homework

58121102 Jiang Yuchu

October 31, 2022

P$_{183}$ 1. Write a C++ function *length* to count the number of nodes in a chain.What is the time complexity of your function?

Answer:

```
size_t length() { // in class Chain
    size_t count = 0;
    for (ChainNode *node = first; node; node = node->link)
        ++count;
    return count;
}
```

It's obviously that the time complexity is $O(n)$, where n is the amount of nodes.

P$_{183}$ 2. Let x be a pointer to an arbitrary node in a chain.Write a C++ function to delete this node from the chain.If $x == first$,then $first$ should be reset to point to the new first node in the chain. What is the time complexity of your function?

Answer:

The following code is the way of forward linked list deletion in the linux core. By using a secondary pointer, the judgment of whether first is null is avoided.

```
void remove(ChainNode* x){ // in class Chain
    ChainNode* node, **curr;
    for (curr = &first; (node = *curr); curr = &node->link;) {
        if (node == x) {
            *next = curr->link;
            delete curr;
            return;
        }
    }
}
```

It's obviously that the time complexity is $O(n)$, where n is the amount of nodes.

P$_{194}$ 3. Write a C++ function to copy the elements of a chain into an array.Use the STL function copy together with array and chain iterators.

Answer:

```
template <class T, class = std::enable_if_t<std::is_array_v<T>>>
void copyToArray(T& arr) const { // in my custom forward linked list
                                 // see slist.h
    copy(begin(), end(), std::begin(arr));
}
```

My custom forward linked list is in *slist.h*. Note that this function was not added to the above file, but after adding, it can be tested well in main function.

P$_{194}$ 4. Let $x_1, x_2, \cdots x_n$ be the elements of a chain. Each $x_i$is an integer.Write a C++ function to compute the expression $\sum_{i=1}^{n-5}(x_i * x_{i+5})$. [Hint: use two iterators to simultaneously traverse the chain.]

Answer:

```cpp
int func() const { // in my custom forward linked list
                   // see slist.h
    if (size() < 5) return NaN;
    auto ia = begin(), ib = std::next(begin(), 5);
    int sum = 0;
    for (; ib != end(); ++ia, ++ib)
        sum += (*ia) * (*ib);
    return sum;
}
```

My custom forward linked list is in *slist.h*. Note that this function was not added to the above file, but after adding, it can be tested well in main function.

P$_{184}$ 6. It is possible to traverse a chain in both directions (i.e., left to right and a restricted right-to-left traversal) by reversing the links during the left-to- right traversal. A possible configuration under this scheme is given in Figure 4.12. Assume that I and r are data members of *Chain*.

Answer:
Sorry, I cannot understand what it means.

P$_{201}$ 2. Develop and test a complete C++ template class for linked queues.

Answer:
See *p201-2.cpp*. For the following test code snippet, it outputs 2 1 3.

```cpp
LinkedQueue<int> l, l2;
l.push_front(1);
l.push_back(2);
l.push_front(3);
l2 = l;
while (!l2.empty()) {
    cout << l2.back() << " ";
    l2.pop_back();
}
```

P$_{225}$ 2. Write a function `void DblList: Concatenate(DblList m)` to concatenate the two lists `*this` and m. On completion of the function, the resulting list should be stored in `*this` and the list m should contain the empty list. Your function must run in O(1)time.

Answer:
This is a doubly linked circular list with header node.

```cpp
void DblList: Concatenate(DblList m) {
    DblListNode* tail = header->left,* other_tail = m.header->left;
    tail->right = m.header->right;
    m.header->right->left = tail;
    other_tail->right = header;
    header->left = other_tail;
    m.header->left = m.header->right = m.header;
}
```

Experiment

$\mathbf{P}_{209}$5.

1 Experiment target

1.1 Background

Develop a C++ class Polynomial to represent and manipulate univariate polynomials with integer coefficients (use circular linked lists with header nodes). Each term of the polynomial will be represented as a node. Thus, a node in this system will have three data members *coef, exp, link*. The external (i.e., for input or output) representation of a univariate polynomial will be assumed to be a sequence of integers of the form: $n, c_1, e_1, c_2, e_2, c_3, e_3, \cdots c_n, e_n$, where e_i; represents an exponent and c_i; a coefficient; n gives the number of terms in the polynomial. The exponents are in decreasing order–$e_1 > e_2 > \cdots > e_n$

1.2 Goals

Write and test the following functions:

(a) `istream& operator>>(istream& is, Polynomial& x)`: Read in an input polynomial and convert it to its circular list representation using a header node.

(b) `ostream& operator<<(ostream& os, Polynomial& x)`: Convert x from its linked list representation to its external representation and output it.

(c) `Polynomial::Polynomial(const Polynomial& a)`[Copy Constructor]: Initialize the polynomial *this to the polynomial a.

(d) `Polynomial& Polynomial::operator=(const Polynomial& a)` [Assignment Operator]: Assign polynomial a to *this.

(e) `Polynomial::~Polynomial()`[Destructor]: Return all nodes of the polynomial *this to the available-space list.

(f) `Polynomial operator+ (const Polynomial& b) const` [Addition]: Create and return the polynomial *this + b.

(g) `Polynomial operator-(const Polynomial& b) const` [Subtraction]: Create and return the polynomial *this - b.

(h) `Polynomial operator*(const Polynomial& b) const` [Multiplication]: Create and return the polynomial *this * b.

(i) `float Polynomial Evaluate(float x) const` : Evaluate the polynomial *this at x and return the result.

2 Basic idea

1. Each polynomial is to be represented as a circular list with header node. To delete polynomials efficiently, we need to use an available-space list and associated functions as described in Section 4.5.

2. Let header be allocated on the stack instead of the heap. This can provide better performance and exception safety at runtime.

3. The header is `NodeBase` which has no specific data member instead of `ListNode`. This is because if we allocate header on the stack, we have to initialize the data member at once. However, the data member might not be default constructible. In fact, we also have another way to avoid this problem(see STL source code), but using `NodeBase` could be more intuitive.

4. Use `std::atexit()` to register callback function to release resource of available-space list(although the OS guarantees that memory will be reclaimed after the program exits, it's always a good practice to do this manually).

5. Using an available-space list is a fantastic idea, but if `value_type` is not copy assignable or move assignable, then this method also fails, then we have to use ordinary way to delete node. We can use select better solution at compile time with tag dispatching
6. For implementing available-space list, I developed a new allocator called `av_allocator<>`(see stuff.h), which is the same programming paradigm as STL.

3 Experimental procedure

I use the following code to test.

```
Polynomial p1;
 cout << "input␣p1(n␣coef1␣exp1␣...␣coefn␣expn)" << endl;
 cin >> p1;
 Polynomial p2;
 cout << "input␣p2(n␣coef1␣exp1␣...␣coefn␣expn)" << endl;
 cin >> p2;
 cout << format("p1␣+␣p2:␣{}\n", p1 + p2);
 cout << format("p1␣-␣p2:␣{}\n", p1 - p2);
 cout << format("p1␣*␣p2:␣{}\n", p1 * p2);
 cout << "evalue␣at␣x0=2␣of␣p1␣is␣" << p1.evaluate(2);
```

4 Results and analysis

After I enter

```
3
1 2 2 3 3 4
2
1 2 3 4
```

The program outputs

```
p1 + p2: 6x^4 + 2x^3 + 2x^2
p1 - p2: 2x^3
p1 * p2: 9x^8 + 6x^7 + 6x^6 + 2x^5 + 1x^4
evaluate at x0=2 of p1 is 68
```