浙江工业大学

C++程序设计实验报告

实验题目　函数、文件、链表

学　　号　302023315399

姓　　名　郑华展

班　　级　软件工程 6 班

提交日期　2024.03.13

一、 实验目的：

1. 掌握函数重载、函数模板的定义和使用
2. 掌握文件的输入输出
3. 掌握单链表的创建、遍历和节点的删除

二、 实验内容：

1．分别用函数重载、函数模板实现：

1）求三个 int 的最大值

2）求三个 double 型数据的最大值

3）求三个 long 型数据的最大值

2. 从键盘输入 5 个同学的信息（包括姓名、性别和成绩），存储到文件 aa.txt 中，然后从文件 aa.txt 中读出成绩大于 60 分的学生信息（包括姓名、性别和成绩）

3. 输入 n 以及 n 个学生的信息（姓名，成绩），再删除其中成绩小于 60 分的节点，然后顺序输出这些学生的信息。要求用链表数据结构完成，并将删除节点的操作编写为函数。

三、实验过程

1. 分别用函数重载、函数模板实现：

1）求三个 int 的最大值

2）求三个 double 型数据的最大值

3）求三个 long 型数据的最大值

（1）程序代码

```
// 函数重载
#include<iostream>
using namespace std;
int max(int a, int b){
    return a>b?a:b;
}
double max(double a, double b){
    return a>b?a:b;
}
long max(long a, long b){
```

```
    return a>b?a:b;
}
int main(){
    int a = 1, b = 2;
    cout<<max(a,b)<<endl;
    double c = 1.0, d = 2.1;
    cout<<max(c,d)<<endl;
    long e = 111111111, f = 22222222222;
    cout<<max(e,f)<<endl;
    return 0;
}


// 函数模板
#include<iostream>
using namespace std;
template<class T>
T maxn(T a, T b){return a>b?a:b;}
int main(){
    int a = 1, b = 2;
    cout<<maxn(a,b)<<endl;
    double c = 1.1, d = 1.2;
    cout<<maxn(c,d)<<endl;
    long e = 111111111, f = 22222222222;
    cout<<maxn(e,f)<<endl;
    return 0;
}
```

（2）运行结果

```
// 函数重载
2
2.1
22222222222
// 函数模板
2
1.2
22222222222
```

（3）结果分析

// 函数重载

这里，我们有三个名为 max 的独立函数，分别专门处理 int、double 和 long 数据类型。在编译过程中，编译器会根据调用中使用的参数类型选择合适的重载函

数。例如，max(a, b) 调用 int max 函数

// 函数模板

函数模板是一种通用函数，可以与各种数据类型 (T) 协同工作，例如上述代码中的 int、double 和 long。调用 maxn(a, b) 时，编译器会根据 a 和 b 的类型自动推导出模板参数 (T) 为 int。同样，它会为其他调用推导出 double 和 long。功能: 该函数简单地比较 a 和 b 并返回较大的值。

// 总结

个人更倾向于使用函数模板，因为这样在实际开发过程中能够省去很多冗余代码，提高代码的复用性，让代码更加简洁、优雅。相应的，我觉得函数重载也会一定程度上让函数更加直观，但是可能也会因此导致产生许多不明确的函数。让重载函数的多义性无法确定，直到遇到函数调用。这就是我不喜欢使用函数重载的原因。两者主要区别在于：函数模板参数的推导是发生在编译的时候，而使用函数重载方法的函数选择是发生在程序运行的时候；函数重载要求参数个数或类型不同，函数模板则要求参数个数必须一样。

2. 从键盘输入 5 个同学的信息（包括姓名、性别和成绩），存储到文件 aa.txt 中，然后从文件 aa.txt 中读出成绩大于 60 分的学生信息（包括姓名、性别和成绩）

（1）程序代码

```
#include<iostream>
using namespace std;
struct node{
    string name,gander;
    int grade;
}mp[10];
int main(){
    freopen("aa.txt","w", stdout);
    for(int i=1;i<=5;i++){
        cin>>mp[i].name>>mp[i].gander>>mp[i].grade;
        if(mp[i].grade > 60) cout<<mp[i].name<<" "<<mp[i].gander<<"
"<<mp[i].grade<<endl;
    }
    fclose(stdout);
    return 0;
}
```

（2）运行结果

Input:

精小弘 女 100

黑白 男 100

目猫 男 60

艾斯比 男 59

鲨鱼 男 0

Output:

// aa.txt

精小弘 女 100

黑白 男 100

目猫 男 60

（3）结果分析

上述程序定义了一个 node 结构体，用来记录学生的相关信息，然后就是用 freopn 重定向输出流到文件。读取的时候直接判断成绩是否达标，然后直接输出.

freopen 函数用于将标准输入/输出流重新定向到文件。相较于其他的文件读写函数，我更喜欢使用 freopen…因为容易写，当年打 oi 的经典（）

freopen 语法如下:

```
FILE *freopen(const char *filename, const char *mode, FILE *stream);
```

filename: 要重定向到的文件的路径名；mode: 打开文件的模式，可以是 "r"（读取）、"w"（写入）、"a"（追加）等；stream: 要重定向的标准流，可以是 stdin（标准输入）、stdout（标准输出）、stderr（标准错误）等。

其实他是有返回值的，成功则返回重定向后的流指针，失败则返回空指针。因此我们可以用一个指针变量去承接返回值，就可以判断文件读写操作是否成功.

3. 输入 n 以及 n 个学生的信息（姓名，成绩），再删除其中成绩小于 60 分的节点，然后顺序输出这些学生的信息。要求用链表数据结构完成，并将删除节点的操作编写为函数。

（1）程序代码

```
#include <iostream>
#include <string>

using namespace std;

struct Student {
    string name;
    int score;
```

```
    Student* next;
};

// 在链表尾部插入节点
void insert(Student*& head, string name, int score) {
    Student* newStudent = new Student;
    newStudent->name = name;
    newStudent->score = score;
    newStudent->next = nullptr;

    if (head == nullptr) {
        head = newStudent;
    } else {
        Student* current = head;
        while (current->next != nullptr) {
            current = current->next;
        }
        current->next = newStudent;
    }
}

// 删除成绩低于 60 分的节点
void deleteBelow60(Student*& head) {
    while (head != nullptr && head->score < 60) {
        Student* temp = head;
        head = head->next;
        delete temp;
    }

    if (head != nullptr) {
        Student* current = head;
        while (current->next != nullptr) {
            if (current->next->score < 60) {
                Student* temp = current->next;
                current->next = current->next->next;
                delete temp;
            } else {
                current = current->next;
            }
        }
    }
}

// 打印链表中的学生信息
```

```
void printList(Student* head) {
    Student* current = head;
    while (current != nullptr) {
        cout << current->name << " "<< current->score << endl;
        current = current->next;
    }
}

int main() {
    int n;
    cin >> n;

    Student* head = nullptr;

    for (int i = 0; i < n; ++i) {
        string name;
        int score;
        cin >> name >> score;
        insert(head, name, score);
    }
    printList(head);

    // 删除成绩低于 60 分的学生节点
    deleteBelow60(head);

    printList(head);

    // 释放链表内存
    Student* current = head;
    while (current != nullptr) {
        Student* temp = current;
        current = current->next;
        delete temp;
    }

    return 0;
}
```

（2）运行结果

Input:

5

精小弘 100

黑白 100

目猫 60

艾斯比 59

鲨鱼 0

Output:

精小弘 100

黑白 100

目猫 60

（3）结果分析

插入函数 insert：

void insert(Student*& head, string name, int score) 函数用于在链表尾部插入新的学生节点。首先创建一个新的学生节点 newStudent，然后将传入的姓名和成绩赋值给该节点。如果链表为空（即 head == nullptr），则将新节点设置为头节点。否则，遍历链表直到找到最后一个节点，然后将新节点连接到最后一个节点的后面。

删除函数 deleteBelow60：

void deleteBelow60(Student*& head) 函数用于删除成绩低于 60 分的学生节点。首先，通过循环删除头节点中成绩低于 60 分的所有节点。然后，从头节点开始遍历链表，如果下一个节点的成绩低于 60 分，则删除该节点；否则，继续遍历下一个节点。

打印函数 printList：

void printList(Student* head) 函数用于打印链表中所有学生节点的信息。

通过循环遍历链表，依次打印每个节点的姓名和成绩。

四、小结与收获

1.函数重载和函数模板的使用：掌握了函数重载和函数模板的定义和使用方法。函数重载可以根据参数的类型和数量定义多个同名函数，提高了代码的灵活性和

可读性；而函数模板则可以编写通用的代码，适用于多种数据类型，提高了代码的复用性和灵活性。

2.文件的输入输出

3.单链表的创建、遍历和节点删除：掌握了单链表的基本操作，包括链表的创建、遍历和节点的删除等。