考试科目名称 高级程序设计 C++ （A 卷）

考试方式：闭卷 考试日期 2020 年 8 月 22 日 教师 郑滔、潘敏学

系（专业） 软件学院（软件工程） 年级 大二 班级

学号 姓名 成绩

题号	一	二	三	四	五	六	七	八	九	十
分数										

注意：所有作答请直接写在卷面上。

得分	

一、简答题（本题满分 44 分，第 1 小题 8 分，其余小题每题 6 分）

1、 请简述 C++程序设计语言的设计理念、演化历程（包括主要的贡献者），并讨论 Simula 67 在其中的作用。

参见历史题？

2、表达式的值有哪些因素决定？表达式会存在副作用吗？

优先级，结合性，求值次序，类型转换

++X X++会产生副作用

3、请解释指针类型和引用类型的差别。

调用方式不同，一个是—>间接访问 一个是.直接访问

安全要求不同
（1）引用被创建的同时必须被初始化（ 指针则可以在任何时候被初始化）。
（2）不能有 NULL 引用，引用必须与合法的 存储单元关联（指针则可以是 NULL）。
（3）一旦引用被初始化，就不能改变引用的关系（指针则可以随时改变所指的对象）

4、简述 C++中继承与虚继承的差异。

虚继承的虚基类会被合并，只有一个副本
虚继承时，不会先构造虚基类的
虚继承更像是一种组合关系，虚继承的基类无法用 static_cast 强制转换成派生类

虚基类的构造函数优先于非虚基类的构造函数
虚基类由最新派生出的类的构造函数调用

5、 为什么要实现两个版本的下标运算符重载？

因为对非常量对象和常量对象的隐含参数 this 类型是不一样的
需要分别对应
const A* const this
A* const this

此外对 const 对象返回 const 值也更符合语义

6、请简述右值引用与左值引用的异同点。

左值引用必须用于左值，右值引用必须用于右值////（这不废话吗= =）
引用建立后，实际上把右值转换成了左值，两者都能进行正常的赋值等有内存变量的举措

7、简述 Lambda 表达式的作用，并比较它与重载了函数调用操作符的函数对象的差别。

实现匿名函数，使函数声明更加简洁
EMMM 不会写

得分	

二、程序理解题（本题满分 36 分，每小题 6 分）

请仔细阅读代码，并判断代码是否存在错误。若不能通过编译，请指出错误代码和错误原因；若能通过编译，请写出代码运行结果。答案直接写在题目右边空白处。注意，本节所有题目的代码，均已#include <iostream>和 using namespace std。

1、

```
typedef double F;
int main()
{
    int a=8,b=3;
    F c=2.0;
    cout << (a/b/c>1?b/a>0?1:2:3);
    return 0;
}
```

Ans :3

2、

```
void func( long a, long b) {
    cout << "long";
}
void func(double a, double b) {
    cout << "double";
}

int main(int argc, const char * argv[]) {
    int a=1, b=1;
    func(a, b);
    return 0;
}
```

编译不通过，两个都是整型提升？

3、

```
class A{
    int x;
public:
    A(inti = 0) {
        x= i;
        cout<< x << " is constructed" << endl;
    }
```

```
    A(A&A) {
        x= A.x;
        cout<< "Copy of " << x<< " is construced." << endl;
    }
    ~A() {
        cout<< x << " is destructed" << endl;
    }
};

int main(intargc, const char* argv[]) {
    Aa(10), b(a);
    return 0;
}
```

10 is constructed

Copy of 10 is construced.

10 is destructed //这个有顺序吗？

10 is destructed

4、

```
int main()
{
    char str1[10]="aaaaaa";
    char str2[10]="aaa";

    char *p1 = str1;
    char *p2 = str2;
    int sum = 0;
    while (*p1 != '\0')
    {
        if (*p1 == *p2)
        {
            p2++;
            int i = 1;
            while (p1[i] != '\0' && p1[i] == *p2)
            {
                i++; p2++;
            }

            if (*p2 == '\0')
            {
                sum++;
                cout << p1 << endl;
            }

            p2 = str2;
        }
        p1++;
    }

    cout << sum << endl;
    return 0;
}
```

Aaaaaa

Aaaaa

Aaaa

aaa

4

Char* 貌似比较到短方结束

5、

```
class FileErrors{  };
class NonExist : public FileErrors {  };

int main() {
    NonExist e;
    try {
        throw e;
    }
    catch (FileErrors& e) {
        cout << "FileErrors" << endl;
    }
    catch (NonExist& e) {
        cout << "NonExist" << endl;
    }
    return 0;
}
```

Ans:FileErrors

6、

```
class Person {
private:
    string name;
public:
    virtual void printName () const {cout << name<<" ";}
    Person(string name) : name(name) {}
};

class Student: public Person{
private:
  string id;
public:
    Student(string id, string name):id(id), Person(name){printName();}
    void printName() const {
        cout << id << " ";
        Person::printName();
    }
};

void Print(Person& p) {p.printName();}

int main() {
    Student s("101", "Tommy");
    Print(s);
    return 0;
}
```

101 Tommy 101 Tommy 好迷，貌似用名空间限定了就不会找虚函数了

得分	

三、 编程题（本题满分 20 分，每小题 10 分）

1、 利用函数模板设计一个求数组元素总和的函数 sum，使得下方的 main 函数可以正确运行。

```
#include <iostream>
using namespace std;

int main ()
{
```

```
    int a[5]={1,2,3,4,5};
    int s1 = sum(a,5);
    cout<< s1<< endl;
    double b[3]={1.5,2.5,3.5};
    double s2 = sum(b,3);
    cout<< s2<< endl;
    return 0;
}
```

```
template <class T>
T sum(T a[] , int size) {
    T sum=a[0];
    for (int i = 1; i < size; i++) {
        sum += a[i];
    }
    return sum;
};
```

2、 编写一个智能指针类 SmartPtr。

1、 实现基于 RAII 的堆上对象资源的管理，即将一个堆上对象资源封装在一个 SmartPtr 对象的生命周期内，避免资源泄漏（4 分）；

2、 实现 SmartPtr 类的解引用运算符（*）和箭头运算符（->）的重载，从而使得 SmartPtr 对象可以像封装在其中的堆上对象的指针一样使用（4 分）；

3、 使用模板类编写 SmartPtr 类，使得它可以封装各种类型的堆上对象资源（2 分）。

```
template<class T>
class SmartPtr {
public:
    SmartPtr(T* p=nullptr):ptr(p){};
    ~SmartPtr() { delete ptr};
    T* operator ->() const{ return ptr};
    T& operator *() const { return *ptr };
private:
    T* ptr;
};
```