序号 ____

2019 学年度秋季学期试卷(A 卷)

课程名: 面向对象程序设计 课程号: 08305121 学分: 5

得分 ____

一、判断题（每小题 2 分，共 20 分）

1. 引用在声明时须对其进行初始化，以绑定某存在的对象。可以声明基类的引用来绑定派生类的对象。 ( )
2. 执行 **delete** p;则删除指针变量 p，故指针变量 p 将不再存在。 ( )
3. **sizeof** 运算符作用于类的对象时，返回该类对象的基本空间所占用的字节数。对象还可能有其资源空间，资源空间的大小依具体对象不同而异。 ( )
4. 构造函数、拷贝构造函数中对缺省冒号语法处理的数据成员，编译系统将自动按该数据成员类型进行默认初始化处理。 ( )
5. 系统并不编译类模板本身，而是编译用具体数据类型实例化后的模板类（如第三大题中的编译 ArrayStack<**int**>及 ArrayStack<**char**>）。 ( )
6. 类的静态数据成员在创建该类的第一个对象时定义并初始化。 ( )
7. 重载后增量运算符时，**operator**++(**int**)中的 **int** 表示需要另外传递一个整型值。 ( )
8. 派生类不继承基类的访问属性为 **private** 的数据成员。 ( )
9. 第三大题中成员函数 **int** ArrayStack<T>::Size() **const**;的隐含的形式参数为 **const** ArrayStack<T> ***const this**。其中，第一个 **const** 表示指针 **this** 所指之处（即本对象）按常量对待。第二个 **const** 表示指针 **this** 为“锁定目标”的指针，其指向不能被改变。 ( )
10. 第四大题中，可以定义 **void** Display(Currency c){}测试对象在函数中值传递。 ( )

得分 ____

二、填空题（每空 2 分，共 20 分）

请根据运行结果，完成程序。

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

void SWAP1R(char &x, char &y)        // 交换字符
{
    ①_____ temp = x;
    x = y;
    y = temp;
}
void SWAP1P(char *x, char *y)        // 交换字符（C 语言解决方案）
{
    ②_____ temp = *x;
    *x = *y;
    *y = temp;
}
void SWAP2R(char *&x, char *&y)      // 交换字符串指针
{
    ③_____ temp = x;
    x = y;
    y = temp;
}
void SWAP2P(char **x, char **y)      // 交换字符串指针（C 语言解决方案）
{
    ④_____ temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
void SWAP_STR(char *const x, char *const y) // 交换字符串内容
{
    char *p = new char[⑤_____+1];
    strcpy(p, x);
    strcpy(x, y);
    strcpy(y, p);
    delete [] p;
}
```

```
int main()
{
   char c1='A', c2='B';
   char *p="Tom", *q="Jerry";       //指向常量区(字符串的内容不能更改)
   char s[10]="Hello",t[10]="World";//数组名为地址常量(即地址不能更改)

   SWAP1R(⑥          );
   cout << "c1 = " << c1 <<", c2 = " << c2 << endl;
   SWAP1P(⑦          );
   cout << "c1 = " << c1 <<", c2 = " << c2 << endl;
   SWAP2R(⑧          );
   cout << "p = " <<p <<", q = " <<q <<endl;
   SWAP2P(⑨          );
   cout << "p = " <<p <<", q = " <<q <<endl;
   SWAP_STR(⑩        );
   cout << "s = " <<s <<", t = " <<t <<endl;
   return 0;
}
```

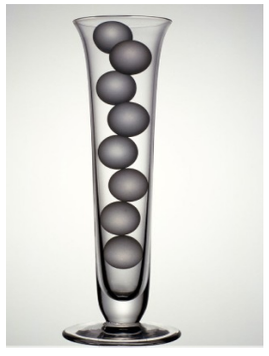
运行结果

```
c1 = B, c2 = A
c1 = A, c2 = B
p = Jerry, q = Tom
p = Tom, q = Jerry
s = World, t = Hello
```

得分 ____

三、阅读程序写出运行结果（每行 1 分，共 30 分）

本题基于类模板 **template** <**typename** T> **class** ArrayStack;该类模板利用数组实现了栈（Stack）这种数据结构。



栈是一种运算受限的线性表。其限制是仅允许在表的一端进行插入和删除运算。这一端被称为栈顶，相对地，把另一端称为栈底。向一个栈插入新元素又称作进栈、入栈或压栈，它是把新元素放到栈顶元素的上面，使之成为新的栈顶元素；从一个栈删除元素又称作出栈或退栈，它是把栈顶元素删除掉，使其相邻的元素成为新的栈顶元素。栈也称为后进先出线性表。

利用数组实现的栈的类模板设计如下。

```
// ArrayStack.h
#ifndef ARRAY_STACK_H
#define ARRAY_STACK_H
#include <iostream>
using namespace std;

#define MAX_STACK 5
template <typename T> class ArrayStack
{
public:
   ArrayStack():top(-1){}          // 构造函数
   int Size() const;               // 获取栈中当前拥有的元素个数
   bool Empty() const;             // 当栈为空时返回 true，否则返回 false
   bool Top(T &t) const;           // 栈不空时返回 true，且通过参数获取栈顶元素；
                                   // 否则返回 false
   bool Push(const T &t);          // 栈未满时将 t 压栈并返回 true,否则返回 false
   bool Pop(T &t);                 // 除完成 Top 的功能外，还使栈顶元素退栈
private:
   T x[MAX_STACK];
   int top;
};
template <typename T> int ArrayStack<T>::Size() const
{
    return top + 1;
}
template <typename T> bool ArrayStack<T>::Empty() const
{
   return top == -1;
}
template <typename T> bool ArrayStack<T>::Top(T &t) const
{
   if(top >= 0)
   {
      t = x[top];
      return true;
   }
   return false;
}
template <typename T> bool ArrayStack<T>::Push(const T &t)
{
   if(top < MAX_STACK-1)
   {
      x[++top] = t;
      return true;
   }
   return false;
}
template <typename T> bool ArrayStack<T>::Pop(T &t)
{
   if(top >= 0)
   {
      t = x[top--];
      return true;
   }
   return false;
}
#endif
```

3.1（10 分）应用测试程序之一（基本测试）

```
#include "ArrayStack.h"

int main()
{
   int x[] = {0, 1, 1, 1, 1, 1, 1, 0, 0, 0};
   int y[] = {47, 26, 71, 38, 69, 12, 67, 99, 35, 94};
   int n = sizeof(x)/sizeof(*x);
   int m = sizeof(y)/sizeof(*y);
   int i, j, z;

   ArrayStack<int> s;   // 创建对象（即建立一个栈）

   for(i=j=0; i<n; i++,j%=m)
   {
      if(x[i]==1)
      {
         if(s.Push(y[j]))     // 请注意 j 的变化规律
            cout << y[j++] << " 入栈" << endl;
         else
            cout << "栈已满，无法入栈" << endl;
      }
      else
      {
         if(s.Pop(z))
            cout << z << " 出栈" << endl;
         else
            cout << "栈已空，无元素出栈" << endl;
      }
   }
   return 0;
}
```

运行结果(3.1)

3.2（8 分）应用测试程序之二（数制转换）

```
#include "ArrayStack.h"

bool Trans(unsigned int n, int Base=2)
{              // 十进制非负整数转换成 Base 进制(Base 可为 2,3,...,36)
   if(n==0)
   {
      cout << 0 << endl;
      return true;
   }
   char BASE[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   int x;
   ArrayStack<int> s;      // 创建对象（即建立一个栈）
   for( ; n!=0; n/=Base)
      if(!s.Push(n%Base))
         return false;
   while(s.Size()>0)
   {
      s.Pop(x);
      cout << BASE[x];
   }
   return true;
}

int main()
{
   int base[] = {2, 3, 4, 5, 8, 10, 16, 36};
   int n = sizeof(base)/sizeof(*base), x=255;

   for(int i=0; i<n; i++)
   {
      cout << '(' << x << ")_10 = (";
      if(!Trans(x, base[i]))
         cout << "栈溢出，无法转换";
      cout << ")_" << base[i] << endl;
   }
   return 0;
}
```

运行结果(3.2) 部分输出结果已给出

(255)_10 = _____

(255)_10 = _____

(255)_10 = _____

(255)_10 = _____

(255)_10 = _____

(255)_10 = _____

(255)_10 = _____

3.3（7 分）应用测试程序之三（括号匹配）

```
#include "ArrayStack.h"

int Pos(char c, const char *str)
{  // 返回字符 c 首次在字符串 str 中出现的位置；若 c 不在字符串 str 中则返回-1
   for(int i=0; str[i]!='\0'; i++)
      if(c == str[i])
         return i;
   return -1;
}
```

```
int Matching(const char *str)
{
   ArrayStack<char> s;
   char c;

   for(int i=0; str[i]!='\0'; i++)
   {
      if(Pos(str[i], "([{")>=0)     // 遇到左括号时的处理：入栈
      {
         if(s.Push(str[i])==false) // 栈溢出
            return -1;
      }
      if(Pos(str[i], ")]}")>=0)     // 遇到右括号时的处理：退栈
      {
         if(s.Pop(c)==false)        // 栈已空
            return -2;
         if(str[i]==')' && c!='(' || str[i]==']' && c!='['
                                  || str[i]=='}' && c!='{')
            return -3;              // 括号不匹配
      }
   }
   return s.Empty()?0:-4;//字符串处理完毕,考虑栈中是否仍留有未匹配的括号
}

int main()
{
   char a[7][100] = {"(())abc{[()]}", "((()))abc{[]}",
                     "((((((((((()", ")(",
                     "(()()abc{[]}", "(())abc{[]()}", "abc" };
   for(int i=0; i<7; i++)
   {
      cout << a[i];
      switch(Matching(a[i]))
      {
      case -1: cout << " 栈空间太小，已溢出，无法判断。" << endl; break;
      case -2: cout << " 栈已空，说明此时右括号多于左括号。" << endl;
                                                              break;
      case -3: cout << " 括号种类不匹配。" << endl;           break;
      case -4: cout << " 字符串处理完毕，但仍有左括号没有匹配完。"<<endl;
                                                              break;
      default: cout << " 括号匹配正确。" << endl;             break;
      }
   }
   return 0;
}
```

运行结果(3.3) 部分输出结果已给出

```
(())abc{[()]} _____
((()))abc{[]} _____
((((((((((() _____
)( _____
(()()abc{[]} _____
(())abc{[]()} _____
abc _____
```

3.4（5 分）应用测试程序之四（将函数入口地址入栈、退栈）

```
#include "ArrayStack.h"
#include <cmath>
double cube(double x)       // 自定义函数，计算参数的立方
{
   return x*x*x;
}
int main()  // 指向函数的指针测试
{
   ArrayStack<double(*)(double)> fs;
   double (*f)(double);
            // 定义指针变量 f，用于指向某一类函数的入口地址。
            // 此处 f 仅能指向“一个 double 形参且返回类型为 double”的函数
   fs.Push(sin);              // 函数名为函数的入口地址
   fs.Push(cos);
   fs.Push(sqrt);
   fs.Push(exp);
   fs.Push(cube);
   fs.Pop(f); cout << f(2) << endl;
   fs.Pop(f); cout << f(1) << endl;
   fs.Pop(f); cout << f(2) << endl;
   fs.Pop(f); cout << f(0) << endl;
   fs.Pop(f); cout << f(0) << endl;
   return 0;
}
```

运行结果(3.4)

_____

_____

_____

_____

_____

得分 | | **四、（10 分）阅读程序，根据指定的输入写出程序运行的结果。**

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

class CNY;                  // 提前声明

class Currency              // 货币类（基类）
{
public:
   Currency(double Value=0,double Rate=1):value(Value),rate(Rate){}
   void SetRate(double Rate){ rate = Rate; }   //调整汇率
   virtual operator CNY() const =0;            //类型转换运算符重载
   virtual void Output(ostream &out) const = 0;
                        // 纯虚函数，输出到形参关联的文件（设备）
   virtual void Input(istream &in) = 0;
                        // 纯虚函数，从参数关联的文件（设备）中读取数据
protected:
   double value, rate;                         // 金额、汇率
};

ostream & operator<<(ostream &out, const Currency &c)
{
   c.Output(out);
   return out;
}
istream & operator>>(istream &in, Currency &c)
{
   c.Input(in);
   return in;
}

class CNY : public Currency         // 人民币类
{
public:
   CNY(double Value=0):Currency(Value,1) {}
   operator CNY() const
   {
      return value;
   }
   void Output(ostream &out) const
   {
      out << "CNY" << value;
   }
   void Input(istream &in)
   {
      cout << "CNY";
      in >> value;
   }
};

class EUR : public Currency         // 欧元类
{
public:
   EUR(double Value=0, double Rate=7.8741):Currency(Value,Rate){}
   operator CNY() const
   {
      return value*rate;
   }
   void Output(ostream &out) const
   {
      out << "EUR" << value;
   }
   void Input(istream &in)
   {
      cout << "EUR";
      in >> value;
   }
};

class GBP : public Currency         // 英镑类
{
public:
   GBP(double Value=0, double Rate=8.9967):Currency(Value,Rate){}
   operator CNY() const
   {
      return value*rate;
   }
   void Output(ostream &out) const
   {
      out << "GBP" << value;
   }
   void Input(istream &in)
   {
      cout << "GBP";
      in >> value;
   }
};
```

```
class JPY : public Currency          // 日元类
{
public:
   JPY(double Value=0, double Rate=0.06116):Currency(Value,Rate){}
   operator CNY() const
   {
      return value*rate;
   }
   void Output(ostream &out) const
   {
      out << "JPY" << value;
   }
   void Input(istream &in)
   {
      cout << "JPY";
      in >> value;
   }
};

class USD : public Currency          // 美元类
{
public:
   USD(double Value=0, double Rate=6.9289):Currency(Value,Rate){}
   operator CNY() const
   {
      return value*rate;
   }
   void Output(ostream &out) const
   {
      out << "USD" << value;
   }
   void Input(istream &in)
   {
      cout << "USD";
      in >> value;
   }
};

void CurrencyExchange(Currency &x)
{
   cout << "今日兑换价: " << x << " = " << CNY(x) << endl;
   cout << "Input your value: ";
   cin >> x;
   cout << x << " = " << CNY(x) << endl;
}
```

```
int main()
{
   CNY c(1);
   EUR e(1);
   JPY j(100);
   GBP g(1);
   USD u(1);

   CurrencyExchange(c);
   CurrencyExchange(e);
   CurrencyExchange(j);
   CurrencyExchange(g);
   CurrencyExchange(u);

   return 0;
}
```

运行结果(4) 部分输出结果已给出，下划线处为键盘输入

```
今日兑换价: CNY1 = CNY_____________________
Input your value: CNY100↵

_____________________________________________
今日兑换价: EUR1 = CNY_____________________
Input your value: EUR100↵

_____________________________________________
今日兑换价: JPY100 = CNY____________________
Input your value: JPY2000↵

_____________________________________________
今日兑换价: GBP1 = CNY_____________________
Input your value: GBP100↵

_____________________________________________
今日兑换价: USD1 = CNY_____________________
Input your value: USD100↵

_____________________________________________
```

得分	

五、（20 分）类模板设计

参照第三大题中的基于数组的栈类模板，设计一个基于动态数组（即堆数组）版的栈模板类 **template** <**typename** T> **class** Stack; 具体要求如下。

① （2 分）类模板基本结构，包括如下数据成员

```
private:
   T *x;            // 栈容器的首地址
   int top, max;    // 栈顶位置（-1 表示栈为空）、栈的容量（创建对象时确定）
```

② （18 分，每个函数 2 分）仿照第三大题设计成员函数，和必要的成员函数。

【注】不必编写测试函数和主函数。