实践5 小结

- 模板的实现。
  - 自定义通用函数模板和类模板时，首先用一个常用类型如char或int定义非通用版本；
  - 非通用版本测试完毕后；
  - 然后增加模板声明前缀，修改代码中相关类型为通用类型名，注意替换不多也不少，完成通用版本的定义；
  - 最后完成通用版本测试。

来自上学期
实践5的小结

实践5 小结

- 循环语句的实现
  - 分析待实现的功能，将需要反复执行的功能，放入循环体内。
  - 考虑循环控制变量和循环体继续执行的条件。
  - 通过循环控制变量和条件的组合，约束循环可以少量反复执行，例如3~5次。测试代码功能，包括循环体内的语句和循环首尾两次执行的条件。
  - 小规模测试无误后，修改循环执行的条件，完成执行完整功能的代码。
- 转向语句
  - **break**语句——在**switch**和循环(**for, while, do-while**)语句中使用，终止当前作用域的程序执行。
  - **continue**语句——仅在循环语句中使用，跳过当前作用域内**continue**语句后的程序的执行。

矩形法（rectangle）求定积分

$$\int_a^b f(x)dx \approx \Delta x(y_0 + y_1 + \cdots + y_{n-1})$$

```
double step = fabs(floor - upper) / n;
double result = fun(floor);
for (int i = 1; i < n; i++)
{
    result += fun(floor + i * step);
}
result *= step;
```

注意是n-1

注意是否存在“差一”误差！

梯形法（ladder）求定积分

$$\int_a^b f(x)dx \approx \frac{\Delta x}{2}(y_0 + 2(y_1 + \cdots + y_{n-1}) + y_n)$$

注意是n

```
double step = (fabs(floor - upper)) / n;
double result = (fun(floor) + fun(upper)) / 2;
for (int i = 1; i < n; i++)
{
    result += fun(floor + i * step);
}
result *= step;
```

注意是否存在“差一”误差！

辛普生法（simpson）求定积分

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}\left(y_0 + y_n + 4(y_1 + y_3 + \cdots + y_{n-1}) + 2(y_2 + y_4 + \cdots + y_{n-2})\right)$$

注意n是偶数

```
double step = fabs(floor - upper) / n;
double result = fun(floor) + fun(upper);
int i;
for (i = 1; i < n; i += 2)
    result += 4 * fun(floor + i * step);
for (i = 2; i < n; i += 2)
    result += 2 * fun(floor + i * step);
result *= step/3;
```

用一个循环时奇偶数判断及对应关系！！！

注意是否存在“差一”误差！

When intervalNum :	10	100	400
********被积函数为sinx *1* *********			
矩形积分法,0~PI/2	0.919403	0.992125	0.998035
梯形积分法,0~PI/2	0.997943	0.999979	0.999999
Simpson积分法,0~PI/2	1	1	1
********被积函数为exp(x) *1.718281828459045***********			
矩形积分法,0~1	1.6338	1.7097	1.71613
梯形积分法,0~1	1.71971	1.7183	1.71828
Simpson积分法,0~1	1.71828	1.71828	1.71828
********被积函数为4.0/(1+x*x) *3.14159265358979********			
矩形积分法,0~1	3.23993	3.15158	3.14409
梯形积分法,0~1	3.13993	3.14158	3.14159
Simpson积分法,0~1	3.14159	3.14159	3.14159

@函数指针法

```
double f3(double x){
    return (1+x+2*x*x+3*x*x*x+4*x*x*x*x);
}
double integer (double (*func)(double),float, float);
int main(){
    double fixint1, fixint2, fixint3;
    fixint1=integer(f1,0.0,3.0);
    fixint2=integer(f2,0.0,3.0);
    fixint3=integer(f3,0.0,3.0);
    cout<<fixint1<<'\n'<<fixint2<<'\n'<<fixint3<<'\n';
    return 0;
}
double integer (double (*func)(double),float a,float b){
    double result,step;
```

指向函数的指针变量定义方式：
返回类型 (*指针变量名)(参数表)
()优先级高于*，不可省略。

@函数模板法

```
class F3 {
public:
    double fun(double x){return (1+x+2*x*x+3*x*x*x+4*x*x*x*x);}
};
template<typename T>double integer(T cf,float a, float b,int n){
    double result,step;
    result=(cf.fun(a)+cf.fun(b))/2;
    step=(b-a)/n;
    for (int i=1;i<n;i++) result+=cf.fun(a+i*step);
    result*=step;
    return result;
}
int main(){
    F1 f1;F2 f2;F3 f3;
    double fixint1, fixint2, fixint3;
    int n=1000;
    fixint1=integer(f1,0.0,3.0,n);
```

编译器根据形参可推断出类型参数

Integer<F1>(f1, 0.0, 3.0, n);

@类模板法

```
class F3 {
public:
    double fun(double x){return (1+x+2*x*x+3*x*x*x+4*x*x*x*x);}
};
template<typename T>class Integer{
    double a,b,step,result;
    int n;                          //分区数量
    T cf;                           //被积函数
public:
    Integer(double aa=0, double bb=0, int nn=100){
        a=aa;    b=bb;    n=nn;
        integerate();
    }
    void putlimits(double aa=0, double bb=0, int nn=100){ //修改积分上下限和分区数
        a=aa;    b=bb;    n=nn;
    }
    void integerate();
    void print(){cout<<"定积分值为： "<<result<<endl;}
};
```

求积分的类模板

模板参数 T 引入被积函数:T为F1,F2,F3.

求积函数为类模板成员函数

面向对象思想

实践7：动态内存分配与深复制

- 要求
  - 理解运行时内存分配的概念，掌握自由存储区内存动态分配的方法 。
  - 理解内部包含动态分配内存的类对象复制时的浅复制和深复制的概念。
  - 会编写深复制复制构造函数、赋值运算符和析构函数。
- 编程：
  - **1.实验18，二-3，为student类编写复制构造函数。**
  - **2. 实验18，二-1，改造mystring类使之能自动适应不同的串长度。**

编程要求（MyString类）

- 参考P226，例7.4对c字符串的处理，完备你自定义的MyString类，主要是三个成员函数的处理，即所谓三规则(rule of three)，大三元(the big three)：
  - （深）拷贝构造函数，包括从C字符串和MyString类的构造
  - （深）赋值运算符“=”，包括从C字符串和MyString类的赋值
  - 析构函数
  - PS:以及所有涉及增或删本对象内容的操作，都要重新申请内存并释放原空间。
- 进一步理解类的封装性(encapsulation)
  - 数据(成员变量)，对数据的操作(成员函数)
  - 运算符重载，提高类的使用体验。
  - 安全的内存资源动态申请、使用和释放。