1.1.1关于数据组织

略

1.1.2关于空间使用

略

1.1.3关于算法效率

例3.写程序计算给定多项式在给定点处的值 $f(x)=a_0+a_1x+\cdots+a_{n-1}x^{n-1}+a_nx^n$

```
double f1(int n,double a[],double x)
{
    int i;
    double p=a[0];
    for(i=1;i<=n;i++)
    {
        p+=(a[i]*pow(x,i));
    }
    return p;
}
```

$f(x)=a_0+x(a_1+x(\cdots(a_{n-1}+x(a_n))\cdots))$

```
double f2(int n,double a[],double x)
{
    int i;
    double p=a[n];
    for(i=n;i>0;i--)
    {
        p=a[i-1]+x*p;
    }
    return p;
}
```

clock（）函数介绍：

```
/*
clock()：捕捉从程序开始运行到clock被调用时所耗费的时间。
单位是clock tick，“时钟打点”。
常数CLK_TCK:机器时钟每秒所走的时钟打点数。
*/

#include <stdio.h>
#include <time.h>
int main()
{
    printf("%d",CLK_TCK);
    return 0;
}

/* 输出结果为1000，即每秒走了1000 */
```

clock（）函数调用：

```c
#include <stdio.h>
#include <time.h>

clock_t start,stop;
/* clock_t是clock()函数返回的变量类型 */

double duration;
/* 记录被测函数的运行时间，以秒为单位 */

int main()
{
    /* 不在测试范围内的准备工作写在clock()调用之前 */
    start=clock();  /* 开始计时 */
    MyFunction();   /* 被测函数放在这里 */
    stop=clock();   /* 停止计时 */
    /* 后继处理卸载最后面，如输出duration的值 */
    return 0;
}
```

例3.写程序计算给定多项式在给定点x=1.1处的值 $f(x)=\Sigma_{i=0}^{9}i\cdot x^i$

计算时间：

```c
#include <stdio.h>
#include <time.h>
#include <math.h>

clock_t start,stop;
double duration;
#define MAXN 10 /* 多项式最大项数，即多项式的阶数+1，常数 */
double f1(int n,double a[],double x);
double f2(int n,double a[],double x);

int main()
{
    int i;
    double a[MAXN]; /* 存储多项式的系数 */
    for(i=0;i<MAXN;i++)
    {
        a[i]=(double)i;
    }

    /* 思考这里两段一样的代码是否可以简化 */

    start=clock();  /* 开始计时 */
    f1(MAXN-1,a,1.1);
    stop=clock();   /* 停止计时 */
    duration=((double)(stop-start))/CLK_TCK;
    printf("ticks1=%f\n",(double)(stop-start));
    printf("duration=%6.2e\n",duration);

    start=clock();  /* 开始计时 */
    f2(MAXN-1,a,1.1);
    stop=clock();   /* 停止计时 */
    duration=((double)(stop-start))/CLK_TCK;
    printf("ticks1=%f\n",(double)(stop-start));
    printf("duration=%6.2e\n",duration);

    return 0;
}

double f1(int n,double a[],double x)
{
    int i;
    double p=a[0];
    for(i=1;i<=n;i++)
    {
        p+=(a[i]*pow(x,i));
    }
    return p;
}

double f2(int n,double a[],double x)
{
    int i;
    double p=a[n];
    for(i=n;i>0;i--)
    {
        p=a[i-1]+x*p;
    }
    return p;
}

/* 运行结果
ticks1=0.000000
duration=0.00e+000
ticks1=0.000000
duration=0.00e+000
 都运行的太快了，不到一个tick
捕捉不到区别
于是考虑让被测函数重复运行多次，测出的总的时钟打点间隔充分长，最后计算被测函数的平均每次运行时间即可
 */
```

计算时间优化：

```c
#include <stdio.h>
#include <time.h>
#include <math.h>

clock_t start,stop;
double duration;
#define MAXN 10     /* 多项式最大项数，即多项式的阶数+1，常数 */
#define MAXK 1e7    /* 被测函数最大重复调用次数 */
double f1(int n,double a[],double x);
double f2(int n,double a[],double x);

int main()
{
    int i;
    double a[MAXN]; /* 存储多项式的系数 */
    for(i=0;i<MAXN;i++)
    {
        a[i]=(double)i;
    }

    /* 思考这里两段一样的代码是否可以简化 */

    start=clock();                                  /* 开始计时 */
    for(i=0;i<MAXK;i++)
    {
        f1(MAXN-1,a,1.1);                           /* 重复调用函数以获得充分多的时钟打点数 */
    }
    stop=clock();                                   /* 停止计时 */
    duration=((double)(stop-start))/CLK_TCK/MAXK;   /* 函数单次运行的时间 */
    printf("ticks1=%f\n",(double)(stop-start));
    printf("duration1=%6.2e\n",duration);

    start=clock();                                  /* 开始计时 */
    for(i=0;i<MAXK;i++)
    {
        f2(MAXN-1,a,1.1);                           /* 重复调用函数以获得充分多的时钟打点数 */
    }
    stop=clock();                                   /* 停止计时 */
    duration=((double)(stop-start))/CLK_TCK;
    printf("ticks2=%f\n",(double)(stop-start));
    printf("duration2=%6.2e\n",duration);

    return 0;
}

double f1(int n,double a[],double x)
{
    int i;
    double p=a[0];
    for(i=1;i<=n;i++)
    {
        p+=(a[i]*pow(x,i));
    }
    return p;
}

double f2(int n,double a[],double x)
{
    int i;
    double p=a[n];
    for(i=n;i>0;i--)
    {
        p=a[i-1]+x*p;
    }
    return p;
}

/* 运行结果
ticks1=886.000000
duration1=8.86e-008
ticks2=87.000000
duration2=8.70e-002
差了一个数量级
运行时间与代码的巧妙程度有关
*/
```

1.1.4抽象数据类型

所以到底什么是数据结构？？？

数据对象在计算机中的组织方式

1. 逻辑结构
    1. 线性结构，一对一
    2. 树形结构，一对多
    3. 图，多对多
2. 物理存储结构
    1. 数组
    2. 链表

数据对象必定与一系列加在其上的操作关联 完成这些操作所用的操作就叫算法

描述数据结构的好办法

抽象数据类型Abstract Data Type

1. 数据类型
    1. 数据对象集
    2. 数据集合相关联的操作集

C语言中独立处理

C++或者JAVA面向对象，把数据对象集和数据集合相关联的操作集封装在一个类

2. 抽象
    1. 与存放数据的机器无关
    2. 与数据存储的物理结构无关
    3. 与实现操作的算法和编程语言均无关

只描述数据对象集和相关操作集“是什么”，并不涉及到“如何做到”的问题

例4.“矩阵”的抽象数据类型定义

类型名称：矩阵（Matrix）

数据对象集：一个 $M*N$的矩阵

$A_{M*N}=(a_{ij})(i=1,\cdots,M;j=1,\cdots,N)$

由 $M*N$个三元组构成，其中 a是矩阵元素的值， i是元素所在的行号， j是元素所在的列号

相关联的操作集有很多很多

操作集：

```
Matrix Create(int M,int N):                 /* 返回一个M*N的空矩阵 */
int GetMaxRow(Matrix A):                    /* 返回A的总行数 */
int GetMaxCol(Matrix A):                    /* 返回A的总列数 */
ElementType GetEntry(Matrix A,int i,int j): /* 返回矩阵A的第i行，第j列的元素 */
Matrix Add(Matrix A,Matrix B):              /* 如果AB的行列数一致，则返回C=A+B，否则返回错误标志 */
Matrix Multiply(Matrix A,Matrix B):         /* 如果A的列数等于B的行数，返回矩阵C=AB，否则返回错误标志 */
```