天津大学

程序设计综合实践课程报告

数据结构实验

学生姓名 陈秋澄

学院名称 智算

专　　业 大类

学　　号 **3022244290**

1. ip 转换

1.1 题目分析

将 32 位二进制数存入 char 数组，之后运用 ASCII 码进行类型转换，最后输出结果，注意格式问题。

1.2 题目代码

```
#include<stdio.h>
#include <iostream>
using namespace std;
int main(){
    char a[32];         /*输入的二进制数据*/
    int b[4]={};        /*输出的 IP 地址数据*/
    int c[8]={128,64,32,16,8,4,2,1};        /*查询表*/
    int temp=0;
    int T;
    cin>>T;
    while(T--){
        for(int i=0;i<32;i++){
        cin>>a[i];
    }
    for(int i=0;i<4;i++){
        int sum=0;
        for(int j=0;j<8;++j){
            temp=8*i +j;
            sum+=(int)(a[temp]-'0')*c[j];  //类型转换
        }
        b[i]=sum;
    }
    cout<<b[0]<<"."<<b[1]<<"."<<b[2]<<"."<<b[3];   //输出结果
    cout<<endl;
    }
}
```

2. 进制转换

2.1 题目分析

首先，当一个正数进行进制转换时，用这个数除以要转换的进制数，保留余数，如果商不为 0，就用商接着除以这个进制数，直到商为 0 时结束，把各个相除的余数存入数组，最后倒叙输出，即为要求进制数。

注意当要转换成 10 到 16 进制时把大于十的数用字母表示。

2.2 题目代码

```
#include<stdio.h>
#include <iostream>
using namespace std;
int main()
{
 int i=0,j,n,b,m,r;
 char a[1000];
 while(cin>>n>>r)
 {
    if (n< 0)         //当负数进行进制转换时进行如下运算
      {
        //printf("-");
        cout<<"-";
        n=n-2*n;
      }
   while(n!=0)
   //当一个正数进行进制转换时，用这个数除以要转换的进制数，保留余数，如果商不
//为 0，就用商接着除以这个进制数，直到商为 0 时结束，把各个相除的余数存入数组，
//最后倒叙输出，即为要求进制数
   {
      ++i;
      b=n/r;            //除以要转换成的进制数
      a[i]=n%r;         //把余数存入数组
      n=b;              //如果商不为 0 就把商接着赋给 n 继续运算
   }
   for(j=i;j>0;j--)
  {
      if(a[j]>=10&&a[j]<=16)
```

```
      {
          if(j!=1)
                      //当要转换成 10 到 16 进制时把大于十的数用字母表示
          cout<<a[j]+55;
          else
          printf("%c\n",a[j]+55);

          i=0;
     }
     else
    {
        if(j!=1)
        printf("%d",a[j]);

        else
        printf("%d\n",a[j]);
        i=0;
    }
  }
 }
 return 0;
}
```

3. 简单计算器

3.1 题目分析

将加减乘除分为四种情况讨论，以含一个运算符的表达式为界，逐式分别进行运算。

发现使用“getchar”效果更好。

3.2 题目代码

```
#include<iostream>
#include<cstdio>
using namespace std;
double num[200];
int main()
{
    double n;
    char s;
    while(cin>>num[0]&&num[0])
    {
        double ans=0.0;
        int i=0;
        s=getchar();
        if(num[0]==0&&s=='\n')
        {
            break;
        }
        while(cin>>s>>n&&s&&n)      //分情况讨论
        {
            if(s=='*')              //情况 1：乘法
            {
                num[i]*=n;
            }
            else if(s=='/')         //情况 2：除法
            {
                num[i]/=n;
            }
            else if(s=='+')         //情况 3：加法
            {
                num[++i]=n;
```

```
            }
            Else                        //情况 4：减法
            {
                num[++i]=-n;
            }
            if(s=getchar()=='\n')
            {
                break;
            }
        }
        while(i>=0)
        {
            ans+=num[i];
            i--;
        }
        printf("%.2f\n",ans);
    }
    return 0;
}
```

4. 队列和栈

4.1 题目分析

总体上，运用两个循环：第一个循环用于判断测试数据次数，第二个循环用于判断每次测试中的步骤数，运用栈相关知识，分别对 pop 操作和 push 操作进行结果的不同表达。

4.2 题目代码

```
#include <iostream>
#include <queue>
#include <stack>
using namespace std;
int s[1000];
char str[10];
int main(){
    int m,n,d,x,flag;
    cin>>m;                 //测试数据次数
    getchar();
    for(int i=1;i<=m;i++){
        queue<int>a;
        stack<int>b;
        d=0;
        flag=0;
        cin>>n;
        for(int i=1;i<=n;i++){   //每次测试中的步骤数
            cin>>str;
            switch(str[1]){
                case 'u':    //判断是 push
                cin>>x;
                a.push(x);
                b.push(x);
                break;
            case 'o':         //判断是 pop
                if(a.empty()||b.empty())
                    flag=1;
                else{
                    a.pop();
                    b.pop();
```

```
                }
                break;
            }

        }
        if(!flag){              //判断是否合法，输出结果
            while(!a.empty()){
                cout<<a.front();
                a.pop();
                if(!a.empty())
                cout<<" ";
            }
            cout<<endl;
            while(!b.empty()){
                s[d]=b.top();
                b.pop();
                d++;
            }
            for(int i=d-1;i>=0;i--){
                cout<<s[i];
                if(i>0)
                    cout<<" ";
            }cout<<endl;
        }
        else{
            cout<<"error"<<endl;
            cout<<"error"<<endl;
        }
    }
    return 0;
}
```

5. 报数

5.1 题目分析

这是约瑟夫环问题，我们可以用链表或数组等多种方式进行求解。
为了更容易理解，我采取数组+循环方式进行求解。另外，本题与第一节课的士兵队列问题有异曲同工之妙。

5.2 题目代码

```
#include <iostream>
#include <cstring>
using namespace std;
int T,n,a[1001];
int main(){
    cin>>T;
    while(T>0){
        memset(a,0,sizeof(a));              //数组初始化
        cin>>n;
        int s=n,k=0;
        while(s>=7){
            for(int i=1;i<=n;i++){
                if(a[i]==0){
                    k++;
                }
                if(k==7){                   //移除倍数为 7 的人
                    k=0;
                    a[i]=1;
                    s--;
                }
            }
        }
        for(int i=1;i<=n;i++){              //输出原始编号
            if(a[i]==0)
                cout<<i<<" ";
        }
        cout<<endl;
        T--;
    }
    return 0;
```

```
}
```

6. 二叉树遍历 1

6.1 题目分析

本程序是读入用户输入的一串先序遍历字符串，根据此字符串建立一个二叉树（以指针方式存储）。

例如如下的先序遍历字符串：

ABC##DE#G##F###

其中“#”表示的是空格，空格字符代表空树。建立起此二叉树以后，再对二叉树进行中序遍历，输出遍历结果。

6.2 题目代码

```
#include<stdio.h>
#include <stdlib.h>
typedef struct node
{
    char ch;
    struct node *lp,*rp;     //左指针和右指针
}node, *tree;

char a[105];
tree t;
int i;
tree xianxu()
{
    tree t;
    if(a[i++] == '#')
    {
        t = NULL;
    }
    else
    {
        t = (struct node *)malloc(sizeof(struct node));   //初始化等
        t->ch = a[i - 1];
        t->lp = xianxu();
        t->rp = xianxu();                          //运用指针
    }
```

```
    return t;
}

void zhonxu_show(tree t)
{
    if(t != NULL)
    {
        zhonxu_show(t->lp);
        printf("%c ",t->ch);
        zhonxu_show(t->rp);

    }
}

void qingchu(struct node *t)      //提高代码稳定性
{
    if(t != NULL)
    {
        qingchu(t->lp);
        qingchu(t->rp);
        free(t);
    }
}
int main()
{
    while(scanf("%s",a) != EOF)
    {
        i = 0;
        t = xianxu();
        zhonxu_show(t);
        printf("\n");
    }
    return 0;
}
```

7. 复原二叉树

7.1 题目分析

我的思路为：前序：DBACEGF，特点：第一个为根节点；中序：ABCDEFG，特点：中间一个为根节点，左边为左子树的节点，右边为右子树的节点。划分：根据前序遍历将更节点将中序序列划分为左子树和右子树。最后再根据中序遍历特点将左子树和右子树划分，知道划分只有一个节点为止

7.2 题目代码

```
#include<bits/stdc++.h>
using namespace std;
//定义二叉树结构
struct Node{
    char data;
    Node *lchild;
    Node *rchild;
};
string pre,in;                 //分别存储前序，中序序列
Node *Create(int preL,int preR,int inL,int inR){
    if(preL>preR)              //当 preL>preR，说明前序序列遍历结束
        return NULL;
    Node *root=new Node;
    root->data=pre[preL];    //根据前序序列特点，首节点为根节点
    //开始遍历中序序列
    int k;
    for(k=inL;k<=inR;k++)
        if(in[k]==pre[preL]) //判断是否在中序序列找到该根节点
            break;
        int numLeft=k-inL;
    root->lchild=Create(preL+1,preL+numLeft,inL,k-1); //递归创建左子树
    root->rchild=Create(preL+numLeft+1,preR,k+1,inR); //递归创建右子树
    return root;
}
      //遍历后序结果
void print_HouXu(Node *root){
    if(root==NULL){
        return;
```

```
    }
    print_HouXu(root->lchild);
    print_HouXu(root->rchild);
    printf("%c",root->data);
}

int main(){
    while(cin>>pre>>in){
        Node *root=Create(0,pre.size()-1,0,in.size()-1);
        print_HouXu(root);
        printf("\n");
    }
    return 0;
}
```

8. 合并果子（堆）

8.1 题目分析

回归问题的本质，我们还是要选取最小的两堆果子，最自然的方式显然是排序了。先排序，选取最小的两堆果子，然后合并，插入。但是插入的效率太低了，应该优化。

我们可以把这些需要插入的点用一个队列存储起来，首先这些需要插入的点肯定会越来越大，显然这相当于延迟插入。当我们目标插入点就是我们当前最小的那一堆的时候，我们就把他插入进来。

代码大概就是，桶排，建立两个队列，排序结果放进第一个当中，合并结果放在第二个当中，每次选从两个队列队头选取比较小的合并。

8.2 题目代码

```
#include <cstdio>
#include <iostream>
#include <queue>
#define int long long
using namespace std;
queue <int> q1;                // 建立两个队列
queue <int> q2;
int to[100005];
void read(int &x){
    int f=1;x=0;char s=getchar();
    while(s<'0'||s>'9'){if(s=='-')f=-1;s=getchar();}
    while(s>='0'&&s<='9'){x=x*10+s-'0';s=getchar();}
    x*=f;
}
signed main() {
    int n;
    read(n);
    for (int i = 1; i <= n; ++i) {
        int a;
        read(a);
        to[a] ++;
    }
```

```
    for (int i = 1; i <= 100000; ++i) {
        while(to[i]) {
            to[i] --;
            q1.push(i);
        }
    }
    int ans = 0;            //建立两个队列，排序结果放进第一个当中，合并结
//果放在第二个当中，每次选从两个队列队头选取比较小的合并
    for (int i = 1; i < n; ++i) {
        int x , y;
        if((q1.front() < q2.front() && !q1.empty()) || q2.empty()) {
            x = q1.front();
            q1.pop();
        }
        else {
            x = q2.front();
            q2.pop();
        }
        if((q1.front() < q2.front() && !q1.empty()) || q2.empty()) {
            y = q1.front();
            q1.pop();
        }
        else {
            y = q2.front();
            q2.pop();
        }
        ans += x + y;
        q2.push(x + y);
    }                           //每次选从两个队列队头选取比较小的合并
    cout<<ans;                 //输出结果
    return 0;
}
```