实验2 程序设计——扑克牌游戏

> 编写bash程序，实现四人打扑克牌比大小游戏。

需求定义

考虑到该题的设计初衷肯定不是考察我们设计规则的能力，又考虑到程序的名字是“比大小游戏”，因此我没有设计太过于复杂的规则，只按照最简单的思路设计了一下规则与流程，如下：

1. 游戏的开头，四个玩家可以自定义游戏ID，否则使用系统的默认名；
2. 每个玩家随机得到13张牌，游戏总共分为13轮，每一轮每个玩家出一张牌，出牌顺序随机；
3. 每一轮根据所出的四张牌的大小次序，按照由大到小的顺序给相应的玩家赋分6、4、2或0；
4. 比较次序的原则是：首先比较大小A>K>Q>J>10>9>8>7>6>5>4>3>2，然后比较花色黑桃>红心>草花>方块；
5. 游戏的最后，总分最高者胜；
6. 当调用程序时，如果输入`--help`参数，则打印帮助信息；

设计

设计思想

考虑到游戏的名字是“比大小游戏”，我也没有接触过太多扑克牌游戏，因此选择了比较简单的游戏规则来实现。

规则已经在上一个模块中说明。此外，为了更贴合四人游戏的设计初衷，每个玩家应该只能看到自己的牌。

功能模块

程序由六个函数组成。

初始化函数`init()`进行游戏的初始化工作。它初始化好一副牌，将花色与大小按顺序组合成一套牌并保存在数组中。其次将每张牌排好顺序，保存在字典中，每张牌对应的权重即是比较大小时的根据。此外，该函数还要处理用户自定义ID，如果用户输入了自定义的ID，要保存好用户的自定义ID。

洗牌函数`shuffle()`将初始化后有序的一套牌打乱，在循环中利用随机数产生下标，将对应的牌与当前的牌交换位置。将牌打乱后，依次将牌复制给各个玩家的数组，即发牌。

打印牌函数`display()`在屏幕上打印各个玩家拥有的手牌。在写这个报告的时候，我又想到其实应该在每一轮每个玩家出牌之前让玩家看到当轮的其他玩家已经出了什么牌，这个是未来改进的地方之一。

比较函数`cmp()`比较根据初始化得到的权重字典，比较两张牌的大小。

出牌函数`play()`获取玩家的输入，为每个玩家进行相应的出牌操作。出过的牌会从每个玩家的手牌数组中删除。

裁判函数`judge()`判定每一轮的得分情况。在写这个函数的时候，我一度纠结于如何进行排序以给分。我的初衷是，数组下标代表了是哪个玩家，数组元素代表了出的是什么牌。如果要排序，那就需要再增加一个数组记录数组下标，否则无法给分；而脚本本身也没有特别好的排序的方法。后来，我想到了其实根本不需要排序，**只需要四张牌两两比较一次，每一次比较给牌面大者赋分**，就不需要排序也可以实现给分。

主函数运行时，首先检查输入的参数及参数数量，在必要时打印帮助信息。如果开始游戏，则初始化、洗牌、利用循环进行十三轮出牌，最后判定胜者。在每一轮游戏中，第一个出牌的玩家是随机决定的，直接将洗牌函数中用到的随机数除以13即可。

数据结构

主要就是运用了数组，程序比较简单，没有使用复杂的数据结构。

算法

实践中发现使用系统变量`$RANDOM`的效果并不理想，因此我利用获取时间的命令简单实现了一个随机数算法。

程序中主要有两处使用了随机数，第一处是初始化中需要洗牌，需要用随机数产生数组下标以打乱顺序；第二处是每一轮的开头需要随机产生出牌顺序。二者只是基数不同，第二处的随机数公式只是第一处的除以十三。

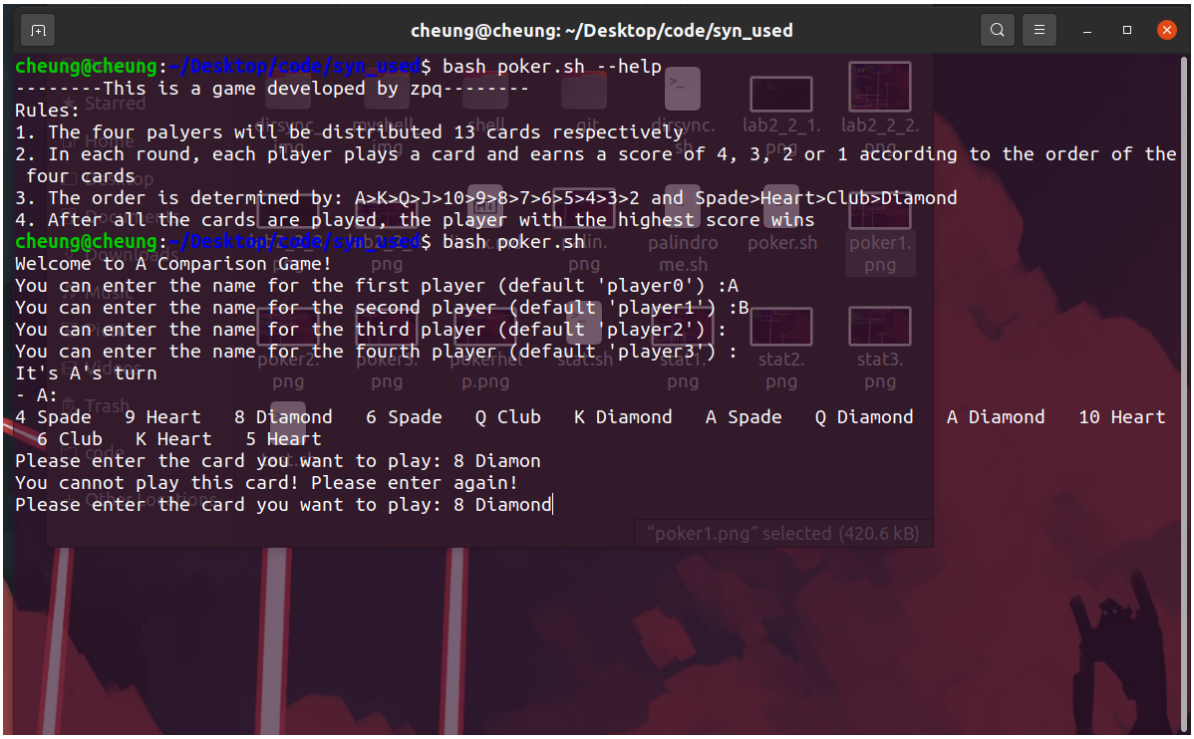
我采用的随机数公式即$random = (A \times B + C) \mod base$的形式。首先调用`date +%N`命令得到纳秒时间戳，然后乘上随机数公式中常用的48271，最后加上一个质数。以shell脚本的形式写出来即是：

```
index=$(date +%N)
index=$(((10#$index*48271+47)%52))
```

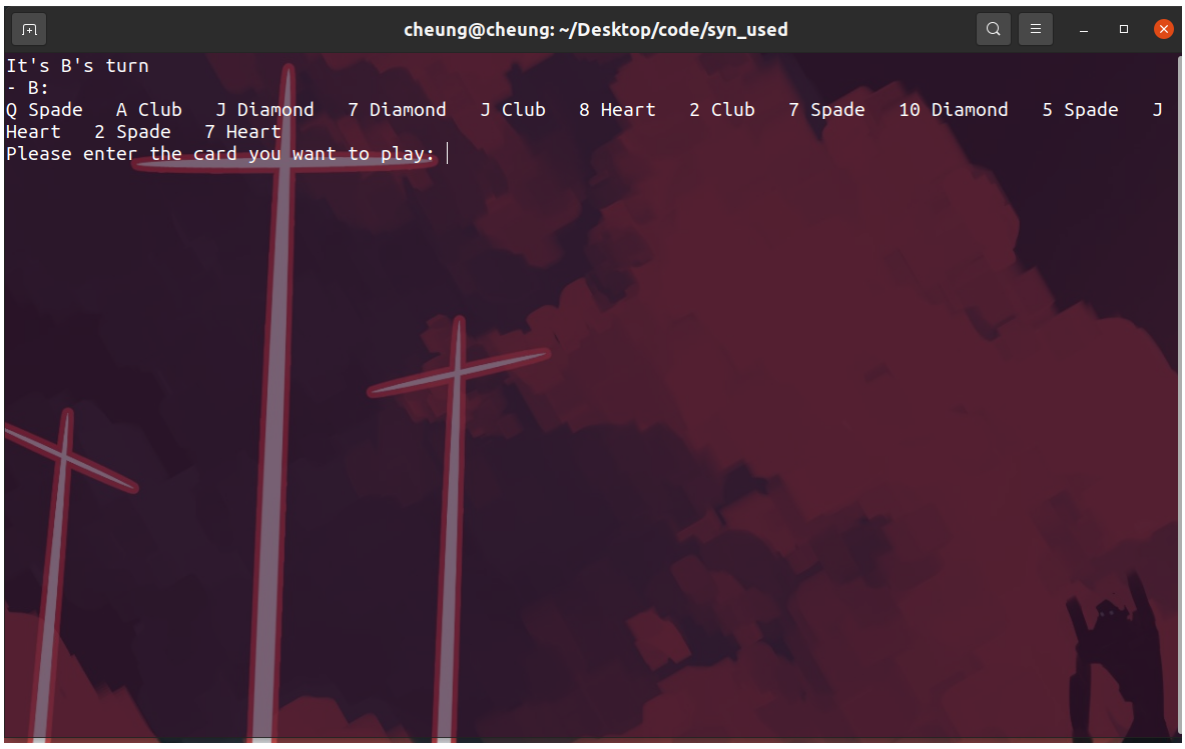
其中需要注意的是，`10#`不可省略。否则，如果得到的时间戳以0开头，系统会认为得到的数是八进制数，很容易出错。`10#index`的作用即是声明这个变量应该被当做十进制数处理。

运行结果

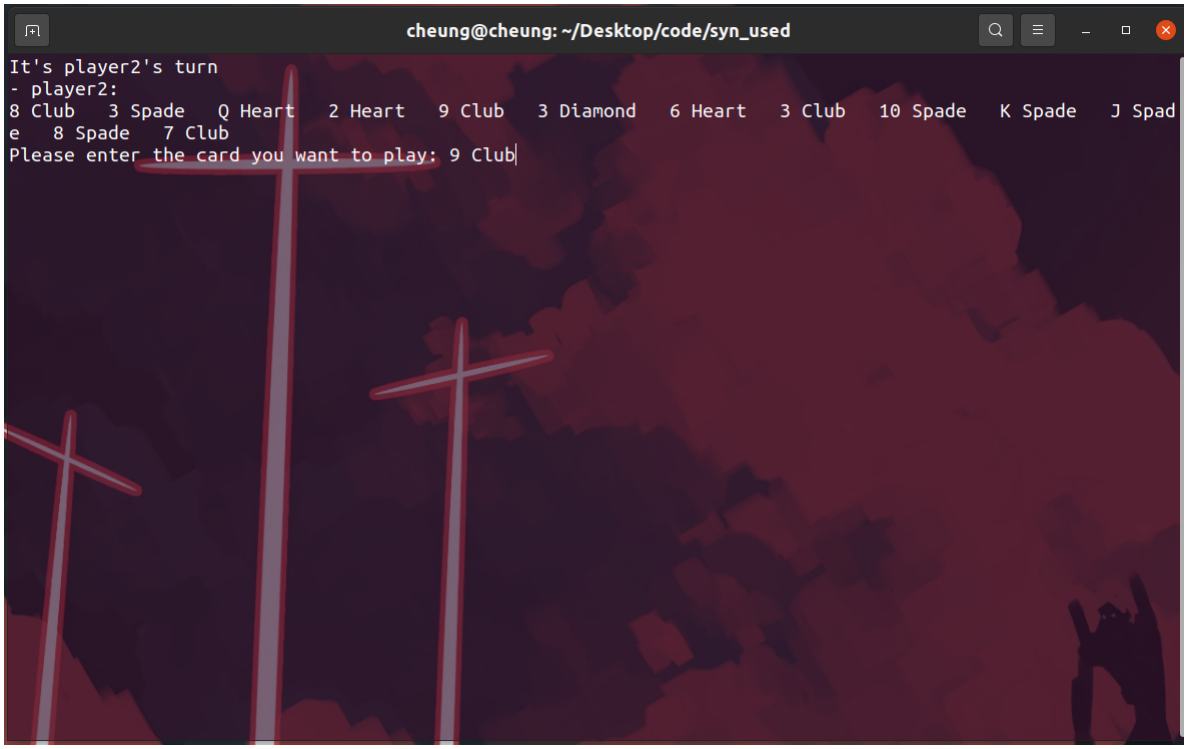
对程序进行一定的测试，首先查看能收正常输出帮助信息，如下图，可见帮助信息正常输出。然后进入游戏，输入两个自定义ID，可见自定义ID生效了，并且如果玩家输入的出牌不对，程序会报错并提示重新输入。

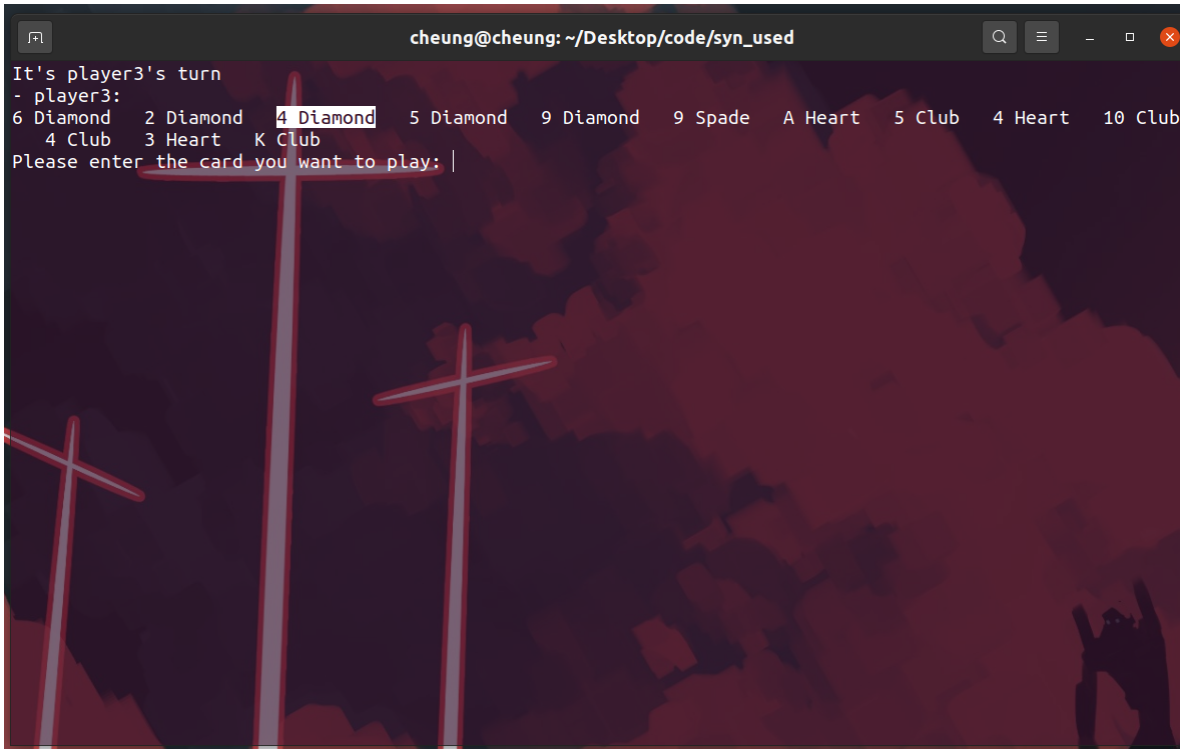


第一个玩家出牌后，可以看到其拥有的牌被隐藏。下一个玩家只能看到自己拥有的牌。

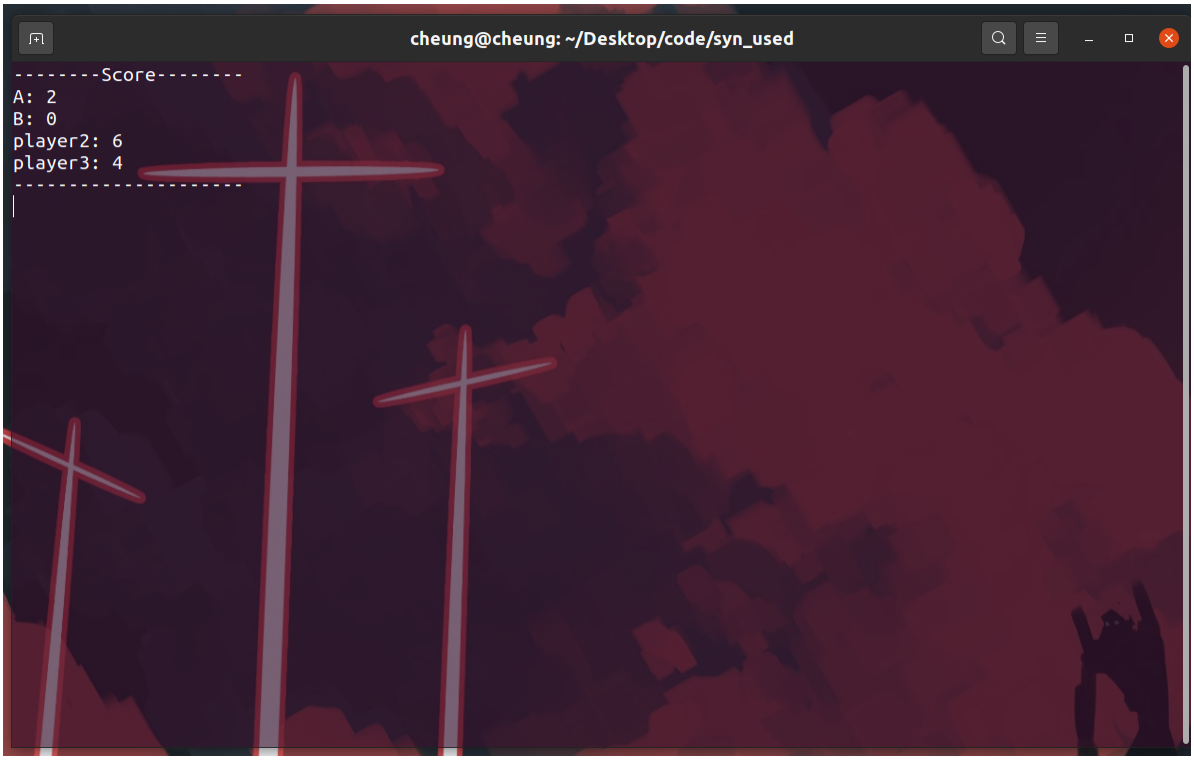


剩下的玩家也一样，并且可以看到这两个玩家没有自定义ID，程序使用了他们的默认ID。

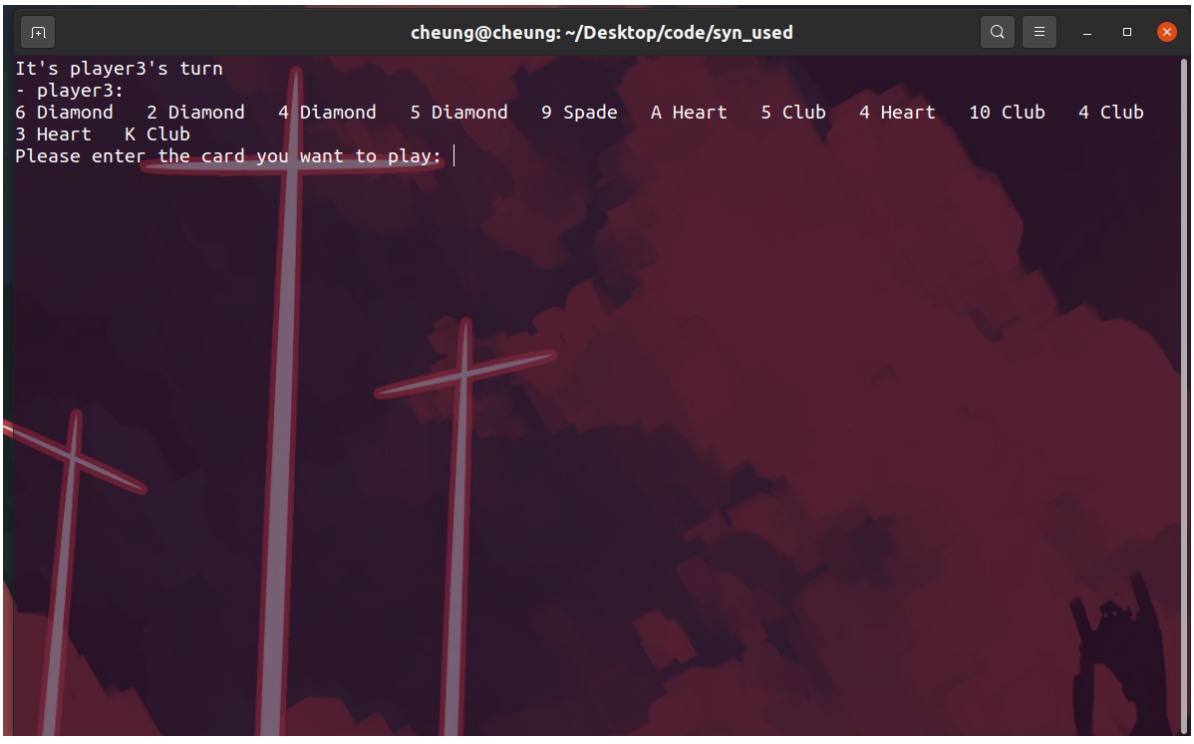




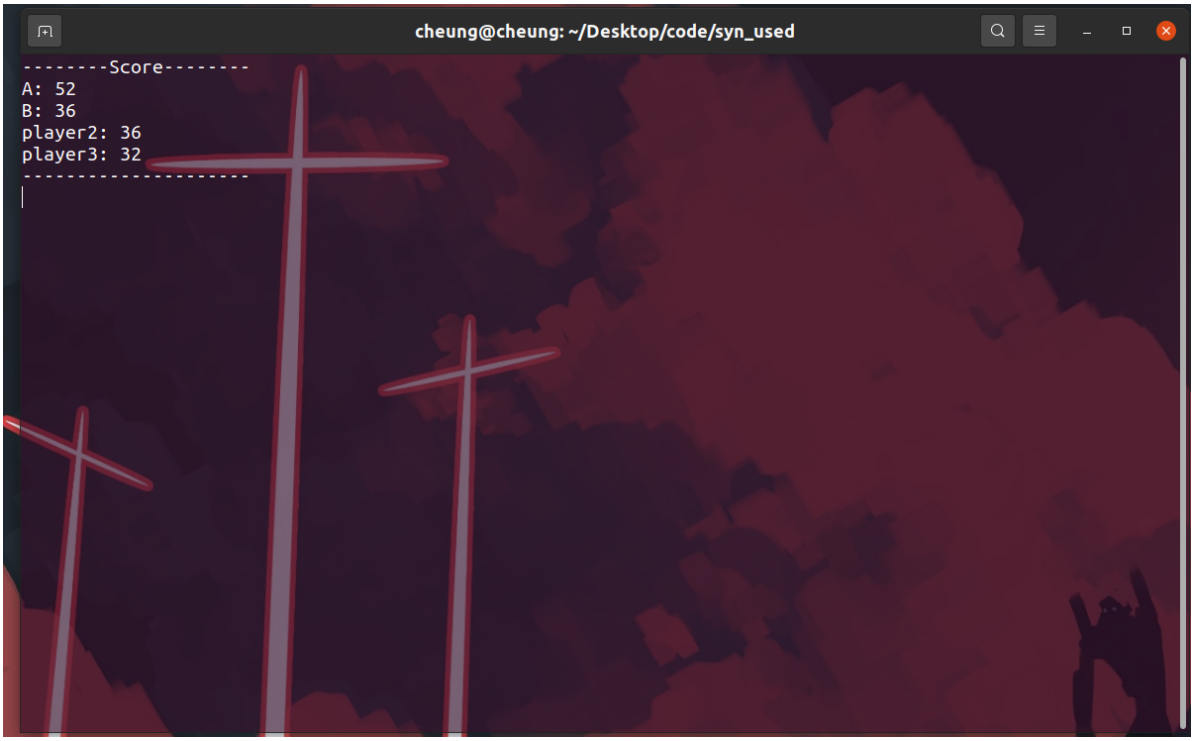
结束一轮游戏后，可以看到这一轮结束后各个玩家的得分情况。



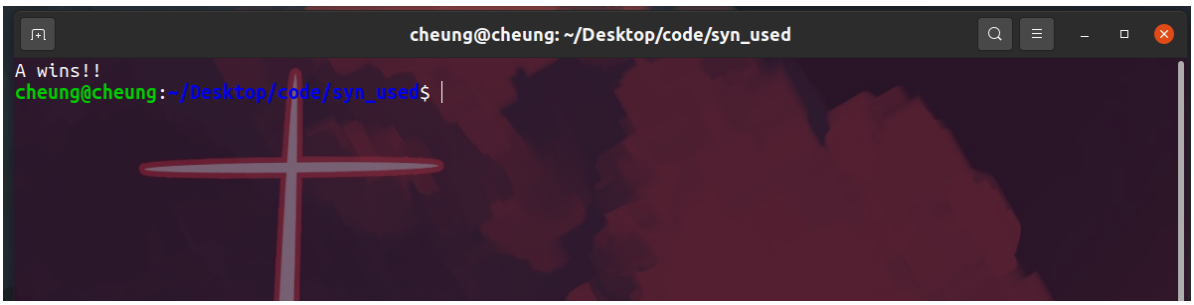
得分展示三秒后清楚，下一轮游戏可以看到第一个出牌的玩家变了，这是随机产生第一个出牌玩家的结果。



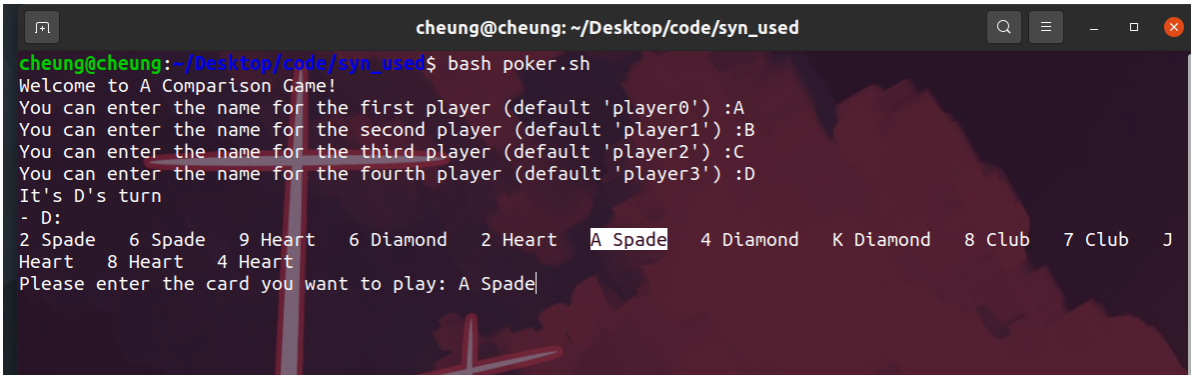
在完成十三轮出牌后，可以看到最终得分情况。

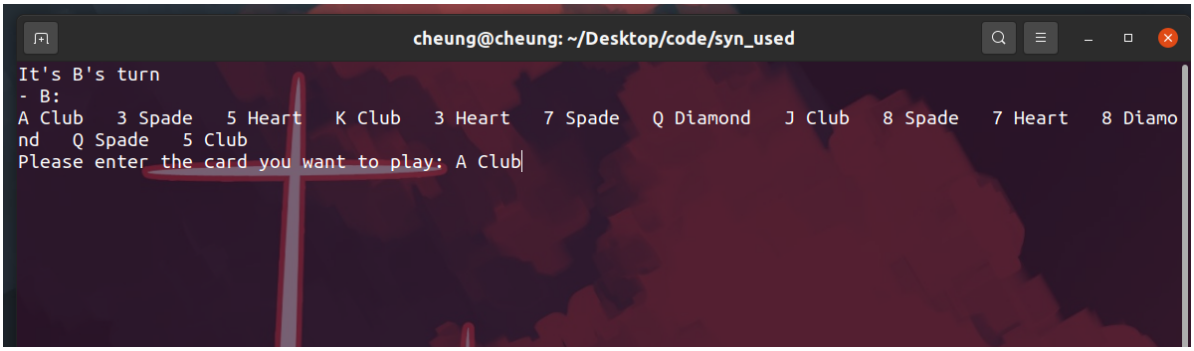


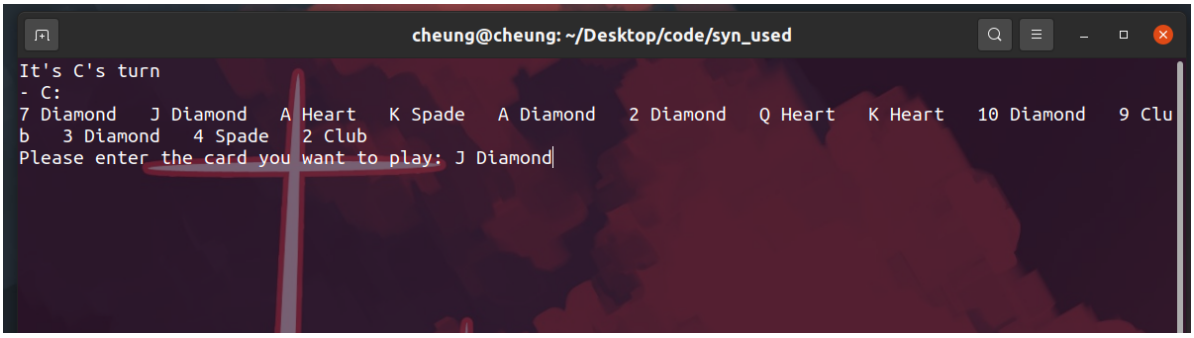
三秒后得分清除，可以看到程序提示得分最高的玩家A获胜。



为了验证自定义的牌面大小顺序是否判定正确，再进行一轮游戏，第一轮各个玩家的出牌如下面几个图所示。







可以看到，出了最大的黑桃A的玩家D得到了6分，其次是同出A的玩家B得到了4分。而出J的两个玩家，花色为黑桃的玩家A得到了两分。由这一系列验证可知，**程序运行与预期相符，结果正确。**

源代码

```
#! /usr/bin/bash
# poker.sh
# 张沛全 3190102214

# 初始化函数
init(){
    # 初始化牌
    for ((i=0;i<4;i++)); do
        for ((j=0;j<13;j++)); do
            cards[$((i*13+j))]="${size[$j]} ${rank[$i]}"
        done
    done
    # 初始化每个牌的大小权重，保存在order数组中，方便比较大小
    for ((i=0;i<13;i++)); do
        for ((j=0;j<4;j++)); do
            index="${size[$i]} ${rank[$j]}"
            order[$index]=$((i*4+j))
        done
    done
```

```
    score[0]=0
    score[1]=0
    score[2]=0
    score[3]=0
    id[0]="player0"
    id[1]="player1"
    id[2]="player2"
    id[3]="player3"
    # 玩家输入id
    for ((i=0;i<4;i++)); do
        if [ $i -eq 0 ]; then
            printf "You can enter the name for the %s player (default
'player%d') :" first 0
        elif [ $i -eq 1 ]; then
            printf "You can enter the name for the %s player (default
'player%d') :" second 1
        elif [ $i -eq 2 ]; then
            printf "You can enter the name for the %s player (default
'player%d') :" third 2
        else
            printf "You can enter the name for the %s player (default
'player%d') :" fourth 3
        fi
        read idinput
        if [[ "$idinput" ]]; then
            id[i]="$idinput"
        fi
    done
    # for ((i=0;i<13;i++)); do
    #     for ((j=0;j<4;j++)); do
    #         index="${size[$i]} ${rank[$j]}"
    #         echo "$index: ${order[$index]}"
    #     done
    # done
}

# 洗牌函数
shuffle(){
    for ((i=0;i<52;i++)); do
        # 生成随机数
        index=$(date +%N)
        index=$(((10#$index*48271+47)%52))
        x=${cards[i]}
        cards[$i]=${cards[$index]}
        cards[$index]=$x
    done
    # 发牌
    for ((i=0;i<13;i++)); do
        cards0[$i]=${cards[i]}
    done
    for ((i=0;i<13;i++)); do
        cards1[$i]=${cards[13+i]}
    done
    for ((i=0;i<13;i++)); do
        cards2[$i]=${cards[26+i]}
    done
    for ((i=0;i<13;i++)); do
        cards3[$i]=${cards[39+i]}
```

```
    done
}

# 打印玩家手头的牌
display(){
    # 打印四个玩家拥有的牌
    if [ $# -eq 0 ]; then
        echo "--------Now each player has--------"
        echo "- ${id[0]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards0[$i]} ]]; then
                echo -n "${cards0[$i]}   "
            fi
        done
        echo ""
        echo "- ${id[1]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards1[$i]} ]]; then
                echo -n "${cards1[$i]}   "
            fi
        done
        echo ""
        echo "- ${id[2]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards2[$i]} ]]; then
                echo -n "${cards2[$i]}   "
            fi
        done
        echo ""
        echo "- ${id[3]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards3[$i]} ]]; then
                echo -n "${cards3[$i]}   "
            fi
        done
        echo ""
        echo "--------done--------"
    elif [ $1 -eq 0 ]; then
        echo "- ${id[0]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards0[$i]} ]]; then
                echo -n "${cards0[$i]}   "
            fi
        done
        echo ""
    elif [ $1 -eq 1 ]; then
        echo "- ${id[1]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards1[$i]} ]]; then
                echo -n "${cards1[$i]}   "
            fi
        done
        echo ""
    elif [ $1 -eq 2 ]; then
        echo "- ${id[2]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards2[$i]} ]]; then
                echo -n "${cards2[$i]}   "
```

```
            fi
        done
        echo ""
    elif [ $1 -eq 3 ]; then
        echo "- ${id[3]}: "
        for ((i=0;i<13;i++)); do
            if [[ ${cards3[$i]} ]]; then
                echo -n "${cards3[$i]}   "
            fi
        done
        echo ""
    fi
    # sleep 10s
}

# 用于比较两个牌的大小，前者大则返回1
cmp(){
    if [ ${order[$1]} -ge ${order[$2]} ]; then
        return 1
    else
        return 0
    fi
}

# 每个玩家出牌的函数，输入参数为玩家的编号
play(){
    echo "It's ${id[$1]}'s turn"
    # 先展示有什么牌
    display $1
    # 出牌
    while [ true ]; do
        # 获取玩家输入
        echo -n "Please enter the card you want to play: "
        read card2play
        if [ $1 -eq 0 ]; then
            # card2play="${cards0[$k]}"
            for ((i=0;i<13;i++)); do
                if [[ ${cards0[$i]} = $card2play ]]; then
                    pool[0]=${cards0[$i]}
                    unset cards0[$i]
                    clear
                    return 0
                fi
            done
            echo "You cannot play this card! Please enter again! "
        elif [ $1 -eq 1 ]; then
            # card2play="${cards1[$k]}"
            for ((i=0;i<13;i++)); do
                if [[ ${cards1[i]} = $card2play ]]; then
                    pool[1]=${cards1[i]}
                    unset cards1[$i]
                    clear
                    return 0
                fi
            done
            echo "You cannot play this card! Please enter again! "
        elif [ $1 -eq 2 ]; then
            # card2play="${cards2[$k]}"
```

```
            for ((i=0;i<13;i++)); do
                if [[ ${cards2[i]} = $card2play ]]; then
                    pool[2]=${cards2[i]}
                    unset cards2[$i]
                    clear
                    return 0
                fi
            done
            echo "You cannot play this card! Please enter again! "
        elif [ $1 -eq 3 ]; then
            # card2play="${cards3[$k]}"
            for ((i=0;i<13;i++)); do
                if [[ ${cards3[i]} = $card2play ]]; then
                    pool[3]=${cards3[i]}
                    unset cards3[$i]
                    clear
                    return 0
                fi
            done
            echo "You cannot play this card! Please enter again! "
        fi
    done
}

# 对于每一张牌，与其后面的牌比较一次，大于则计一分，这样则可以利用下标来记录总得分
judge(){
    # 计分
    for ((i=0;i<3;i++)); do
        for ((j=i+1;j<4;j++)); do
            cmp "${pool[$i]}" "${pool[$j]}"
            if [ $? -eq 1 ]; then
                let "score[$i]+=2"
            else
                let "score[$j]+=2"
            fi
        done
    done
}


#############主函数#############
# 检查传入参数的数量
if [ $# -eq 1 ]; then
    # echo "$1"
    if [ $1 = "--help" ]; then
        echo "--------This is a game developed by zpq--------"
        echo "Rules: "
        echo "1. The four palyers will be distributed 13 cards respectively"
        echo "2. In each round, each player plays a card and earns a score of 6,
4, 2 or 0 according to the order of the four cards"
        echo "3. The order is determined by: A>K>Q>J>10>9>8>7>6>5>4>3>2 and
Spade>Heart>Club>Diamond"
        echo "4. After all the cards are played, the player with the highest
score wins"
        exit 0
    else
        echo "Illegal varibale!"
        exit 1
```

```
    fi
elif [ $# -ne 0 ]; then
    echo "The number of variabls is wrong!"
    exit 1
fi

# 打印信息
echo "Welcome to A Comparison Game!"
# 声明
declare -a cards
declare -a cards0
declare -a cards1
declare -a cards2
declare -a cards3
declare -a pool
declare -a score
declare -a id
declare -A order
# 考虑加入自定义玩家名
# 初始化牌的花色与大小
rank=("Diamond" "Club" "Heart" "Spade")
size=(2 3 4 5 6 7 8 9 10 J Q K A)
init
shuffle
# 出牌13轮
for ((k=0;k<13;k++)); do
    index=$(date +%N)
    index=$(((10#$index*48271+47)%52/13))
    play $index
    index=$(($index+1)%4))
    play $index
    index=$(($index+1)%4))
    play $index
    index=$(($index+1)%4))
    play $index
    index=$(($index+1)%4))
    judge
    echo "--------Score--------"
    echo "${id[0]}: ${score[0]}"
    echo "${id[1]}: ${score[1]}"
    echo "${id[2]}: ${score[2]}"
    echo "${id[3]}: ${score[3]}"
    echo "---------------------"
    sleep 3s
    clear
done
# echo "0: ${score[0]}"
# echo "1: ${score[1]}"
# echo "2: ${score[2]}"
# echo "3: ${score[3]}"
# 获取结果并输出
if [ ${score[0]} -gt ${score[1]} ] && [ ${score[0]} -gt ${score[2]} ] && [
${score[0]} -gt ${score[3]} ]; then
    echo "${id[0]} wins!!"
elif [ ${score[1]} -gt ${score[0]} ] && [ ${score[1]} -gt ${score[2]} ] && [
${score[1]} -gt ${score[3]} ]; then
    echo "${id[1]} wins!!"
```

```
elif [ ${score[2]} -gt ${score[0]} ] && [ ${score[2]} -gt ${score[1]} ] && [
${score[2]} -gt ${score[3]} ]; then
    echo "${id[2]} wins!!"
elif [ ${score[3]} -gt ${score[0]} ] && [ ${score[3]} -gt ${score[2]} ] && [
${score[3]} -gt ${score[2]} ]; then
    echo "${id[3]} wins!!"
else
    echo "There is no winners!!"
fi
exit 0
```