第 1 章　循环与递归

1. 用循环算法找出 5 个自然数中取 3 个数的组合。

C/C++代码：

```
#include <stdio.h>
void combloop1(int n, int r)
{
    for(int i=1; i<=n-2; i++){
        for(int j=i+1; j<=n-1; j++){
            for(int k=j+1; k<=n; k++){
                printf("% d % d % d\n", i, j, k);
            }
        }
    }
}
int main()
{
    int n, r;
    scanf("% d % d", &n, &r);
    combloop1(n, r);
    return 0;
}
```

Python 代码：

```
def combloop1(n, r):
    for i in range(1, n - 1):
        for j in range(i + 1, n):
            for k in range(j + 1, n + 1):
                print(i, j, k)
def main():
   n, r = map(int, input().split())
   combloop1(n, r)

if __name__ == "__main__":
    main()
```

2. 用递归算法找出 5 个自然数中取 3 个数的组合。

C/C++代码：

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

```
//递归函数，用于生成组合
void combine(int start, int n, int r, int index, int combination[]) {
    // 如果组合中的元素数量达到 r，输出当前组合
    if (index == r) {
        for (int i = 0; i < r; i++) {
            printf("%d ", combination[i]);
        }
        printf("\n");
        return;
    }
    // 递归生成组合
    for (int i = start; i <= n; i++) {
        combination[index] = i; // 将当前数字加入组合
        combine(i +1, n, r, index + 1, combination);   // 递归处理下一个数字
    }
}

int main() {
    int n, r;
    scanf("%d %d", &n, &r);
    int combination[r]; // 用于存储当前组合的数组
    // 调用递归函数生成组合
    combine(1, n, r, 0, combination);
    return 0;
}
```

Python 代码：

```
#递归函数，用于生成组合
def combine(start, n, r, index, combination):
    # 如果组合中的元素数量达到 r，输出当前组合
    if index == r:
        print(' '.join(map(str, combination)))
        return
    # 递归生成组合
    for i in range(start, n + 1):
        combination[index] = i# 将当前数字加入组合
        combine(i +1, n, r, index + 1, combination)  # 递归处理下一个数字
def main():
    # 输入 n 和 r 的值
    n, r =map(int, input().split())
```

```
    combination = [0] *  r  # 用于存储当前组合的数组
    # 调用递归函数生成组合
    combine(1, n, r, 0, combination)
if __name__ == "__main__":
        main()
```

3. 用循环和递归算法求 n 的阶乘 $n!$。

C/C++代码：

```
#include <stdio.h>

float fac_recursion(int n){
    if(n==1)
        return 1;
    return n* fac_recursion(n-1);
}
float fac_loop(int n){
    int sum =1;
    while(n>=1){
        sum = sum *  n;
        n--;
    }
    return sum;
}
int main(){
    int n;
    float y;
    scanf("% d", &n);
    y=fac_recursion(n);
    printf("循环算法: % d! = % .0f \n", n, y);
    y=fac_loop(n);
    printf("递归算法: % d! = % .0f \n", n, y);
return 0;
}
```

Python 代码：

```
def fac_recursion(n):
    if n == 1:
        return 1
    return n *  fac_recursion(n - 1)

def fac_loop(n):
```

```
    sum = 1
    while n >= 1:
        sum * = n
        n -=1
    return sum

def main():
    n =int(input())
    y = fac_recursion(n)
    print(f"递归算法: {n}! = {y: .0f}")
    y = fac_loop(n)
    print(f"循环算法: {n}! = {y: .0f}")

if __name__ == "__main__":
    main()
```

4. 用循环和递归算法求斐波那契数列的前 10 项。

C/C++代码：

```
#include <stdio.h>
int fibo_recur(int n)
{
    if(n==1 ||n==2){
        return 1;
    }
    return fibo_recur(n-1)+fibo_recur(n-2);
}

int fibo_loop(int n)
{
    if (n==1 ||n==2){
        return 1;
    }
    int a=1, b=1, t;
    for(int i=3; i<=n; i++){
        t = b;
        b = a+b;
        a = t;
    }
    return b;
}
```

```
int main()
{
    int n, y, i;
    scanf("% d", &n);
    printf("循环算法: ");
    for(i=1; i<=n; i++)
    {
      y=fibo_loop(i);
      printf("% d ", y);
    }
    printf("\n 递归算法: ");
    for(i=1; i<=n; i++)
    {
      y=fibo_recur(i);
      printf("% d ", y);
    }
    return 0;
}
```

Python 代码：

```
#递归实现斐波那契数列
def fibo_recur(n):
    if n == 1 or n == 2:
        return 1
    return fibo_recur(n - 1) + fibo_recur(n - 2)

#循环实现斐波那契数列
def fibo_loop(n):
    if n == 1 or n == 2:
        return 1
    a, b =1, 1
    for i in range(3, n + 1):
        a, b = b, a + b
    return b

def main():
    n =int(input())
    print("循环算法: ", end="")
    for i in range(1, n + 1):
        y = fibo_loop(i)
```

```
        print(y, end=" ")
    print("\n 递归算法: ", end="")
    for i in range(1, n + 1):
        y = fibo_recur(i)
        print(y, end=" ")

if __name__ == "__main__":
    main()
```