

# 全排列生成算法

class project 2017

## 编译和运行

编译运行环境为GCC，Win10，经测试在Ubuntu和Mac系统中也能正常编译运行。目录中的可执行程序是在Win10中生成的，若在你的操作系统中无法运行，请使用make重新编译生成。

### 1. 编译

执行以下命令得到可执行程序main：

```
cd $(path)
make clean
make
```

### 2. 运行

运行参数如下：

```
#!/main -h
Usage: ./main [options]
Options:
  -n INT      size of the full permutations
  -a INT      type of algorithm to generate permutations
               0: dict          字典序
               1: increase      递增进制制
               2: decrease      递减进制制
               3: neighbor      邻位对换
               4: binary_dict (n <= 32) 等价于字典序法的二进制变换算法
               5: binary_new (n <= 32)  新的二进制变换算法
  -h          this help message      帮助信息
```

## 目录结构

```
.
├── binary.h
├── dict.h
├── increase_decrease.h
├── neighbor.h
├── permutation.h
├── main.c
├── Makefile
└── main
```

## 文件说明

### 0. 基于二进制变换规则的全排列生成算法 binary.h

```
// 等价于字典序法的二进制变换规则
count=generate_permutations_by_binary_dict(n);
printf("[+]Binary_Dict,Permutations(%d)=%ld\n",n,count);

// 新的二进制变换规则
count=generate_permutations_by_binary_new(n);
printf("[+]Binary_New,Permutations(%d)=%ld\n",n,count);
```

## 1. 字典序全排列生成算法 dict.h

```
count = generate_permutations_by_dict(n);  
printf("[+]Dictionary,Permutations(%d)=%ld\n",n,count);
```

## 2. 递增 / 减进制全排列生成算法 increase\_decrease.h

```
//递增进制生成算法  
count=generate_permutations_by_increase(n);  
printf("[+]Increase,Permutations(%d)=%ld\n",n,count);  
  
//递减进制生成算法  
count=generate_permutations_by_decrease(n);  
printf("[+]Decrease,Permutations(%d)=%ld\n",n,count);
```

## 3. 邻位对换全排列生成算法 neighbor.h

```
count=generate_permutations_by_neighbor(n);  
printf("[+]Neighbor,Permutations(%d)=%ld\n",n,count);
```

## 4. 其他文件

permutation.h	包含上述算法.h文件，提供应用程序调用
main.c	调用几种全排列生成算法
Makefile	程序编译
main	可执行程序