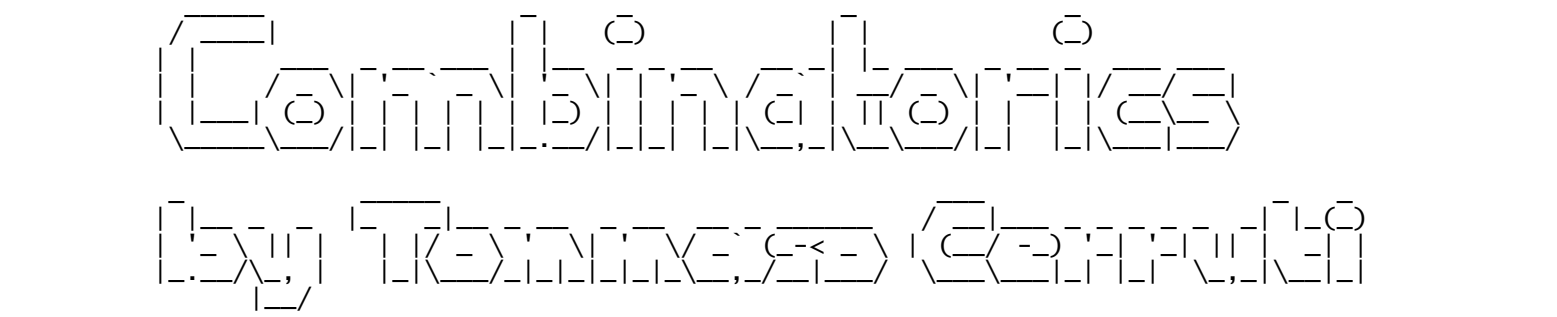

---0-INDICES

There are seven different functions to generate these samples of subsets:
1- Multiplication principle --> multiple sets (choose one of each)
2- Simple arrangements --> ordered subset with no rep
3- Arrangements with repetitions --> ordered substet with rep
4- Simple permutations --> reordering the set (ordered set without rep)
5- Permutations with repetitions --> reodering the set with repeting el.
(ordered set with rep)
6- Simple combinations --> unordered subset without rep
7- Combinations with repetitions --> unordered subset with rep
(Note: there is no unordered set because it would be the set itself)

---1-MULTIPLICATION-PRINCIPLE

Given n sets Si, each one of cardinality |Si|:
How many ordered tuples we can extract?
MS1,...,S(n-1)=product(Si)(0to(N-1))

In practice: When you need to choose one of each set

Example: Any customer can choose 1 apetizer (out of 2), 1 first course (out of
3), and 1 second course (out of 2).
M=2*3*2=12 menus

Implementation (data structure):
```
typedef struct val_s{
    int num_choice;
    int *choices;
} val_t;
val=malloc(n*sizeof(val_t));
for (i=0;i<n;i++)
    val[i].choices=malloc(val[i].n_choice*sizeof(int));
//val[i] storing the set i: how many values(num_choices) and the
values(*choices)
sol=malloc(n*sizeof(int));
//to store the solution
```

Implementation (algorithm):
```
int mult_princ (val_t *val, int *sol, int n, int count, int pos){
    int i;
    if (pos>=n){
        printf("Solution %d: ,count);
        for (i=0;i<n;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0;i<val[pos].num_choice;i++){
        sol[pos]=val[pos].choices[i];
        count=mult_princ(val,sol,n,count,pos+1);
```

```
    }
    return count;
}
```

Example:
```
1 1 1
1 1 2
1 2 1
1 2 2
1 3 1
1 3 2
2 1 1
2 1 2
2 2 1
2 2 2
2 3 1
2 3 2
```

---2-SIMPLE-ARRANGEMENT

Given one set, simple arrangement of n distinct objects of class k (k<=n), is an ordered subset composed of k objects, each can be chosen once
Dn,k=n*(n-1)*...*(n-k+1)=n!/(n-k)!

In practice: When you need to chose all ordered subset of a set of size k (but once you choose the element i you cannot rechoose it --> simple)

Example: How many strings of 2 chars can be formed from the vowels (5)
D=5!/(5-2)!=5*4=20 strings

Implementation (data structure):
```
val=malloc(n*sizeof(int));
//the set
mark=malloc(n*sizeof(int));
//a flag array to check which values are already taken: mark[i]=0->i not taken/mark[i]=1->i taken
sol=malloc(k*sizeof(int));
//solution
```

Implementation (algorithm):
```
int arr(int *val, int *sol, int *mark, int n, int k, int count, int pos){
    int i;
    if (pos>=k){
        printf("Solution %d: ,count);
        for (i=0;i<k;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0;i<n;i++){
        if (mark[i]==0){ //if not marked, i.e. chosen
            mark[i]=1; //mark and choose the next
            sol[pos]=val[i];
            count=arr(val,sol,mark,n,k,count,pos+1); //recur
            mark[i]=0; //unmark, i.e. traceback
        }
    }
    return count;
}
```

Example:
```
(AEIOU)
AE
AI
AO
AU
EA
```

```
EI
EO
EU
IA
IE
IO
IU
OA
OE
OI
OU
UA
UE
UI
UO
```

---3-ARRANGEMENT-WITH-REPETITIONS

Given one set, arrangement with repetitions of n distinct objects of class k (k<=n), is an ordered subset composed of k objects, each can be chosen even more than once
D'n,k=n*n*...*n=n^k

In practice: When you need to chose all ordered subset of a set of size k (and you can rechoose the element i)

Example: How many strings of 2 chars can be formed from the vowels (5)
D=5^2=25 strings

Implementation (data structure):

```
//since repetitions, mark is omitted
val=malloc(n*sizeof(int));
//the set
sol=malloc(k*sizeof(int));
//solution
```

Implementation (algorithm):

```
int arr_rep(int *val, int *sol, int n, int k, int count, int pos){
    int i;
    if (pos>=k){
        printf("Solution %d: ,count);
        for (i=0;i<k;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0;i<n;i++){//same as before but no mark procedure
        sol[pos]=val[i];
        count=arr_rep(val,sol,n,k,count,pos+1);
    }
    return count;
}
```

Example:

```
(AEIOU)
AA
AE
AI
AO
AU
EA
EE
EI
EO
EU
IA
IE
```

```
II
IO
IU
OA
OE
OI
OO
OU
UA
UE
UI
UO
UU
```

---4-SIMPLE-PERMUTATION

Given one set, simple arrangement of n distinct objects of class n, is a simple permutation
Pn=Dn,n=n*(n-1)*...*(n-n+1)=n!

In practice: When you choose all the possible orderings of a set

Example: How many anagrams of the string ORA, i.e. permutations
P=3!=6 strings

Implementation (data structure):
val=malloc(n*sizeof(int));
//the set
mark=malloc(n*sizeof(int));
//a flag array to check which values are already taken: mark[i]=0->i not taken/mark[i]=1->i taken
sol=malloc(n*sizeof(int));
//solution

Implementation (algorithm):

```
//just like simple arrangements, but k=n, so it is not taken into consideration
int perm(int *val, int *sol, int *mark, int n, int count, int pos){
    int i;
    if (pos>=n){
        printf("Solution %d: ,count);
        for (i=0;i<n;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0;i<n;i++){
        if (mark[i]==0){ //if not marked, i.e. chosen
            mark[i]=1; //mark and choose the next
            sol[pos]=val[i];
            count=perm(val,sol,mark,n,count,pos+1); //recur
            mark[i]=0; //unmark, i.e. traceback
        }
    }
    return count;
}
```

Example:
```
ORA
OAR
ROA
RAO
AOR
ARO
```

---5-PERMUTATION-WITH-REPETITIONS

Given one set, arrangement of n objects (of which a,b,c,... are identical) of class n, is a permutation with repetitions
Pn(a,b,c,...)=n!/(a!b!c!..)

In practice: When you choose all the possible orderings of a set but some elements are identical

Example: How many anagrams of the string ORO (note: 2 identical chars)
P=3!/2!=3 strings

Implementation (data structure):
```
//note: n(cardinality of multiset) -> n_dist(cardinality of distinct elements)
val_dist=malloc(n_dist*sizeof(int));
//the set of only distinct objects
mark=malloc(n_dist*sizeof(int));
//a flag array to check which values are already taken: mark[i]>0->i not taken (since could be more than one)/mark[i]=0->i taken
//mark[i]=x, where x is the number of repetitions that i can still take, generally it is 1, but if repetitions (a times), it is a
sol=malloc(n*sizeof(int));
//solution
```

Implementation (algorithm):
```
//just like simple arrangements, but k=n, and we are selecting from the distinct values of n
int perm_rep(int *val_dist, int *sol, int *mark, int n, int n_dist, int count, int pos){
    int i;
    if (pos>=n){
        printf("Solution %d: ,count);
        for (i=0;i<n;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=0;i<n_dist;i++){
        if (mark[i]>0){ //if not empty choose, i.e. chosen all the times
            mark[i]--; //decrement and choose the next
            sol[pos]=val_dist[i];
            count=perm_rep(val_dist,sol,mark,n,n_dist,count,pos+1); //recur
            mark[i]++; //increment, i.e. traceback
        }
    }
    return count;
}
```

Example:
ARA
RAA
AAR

---6-SIMPLE-COMBINATIONS

Given one set, simple combinations of n distinct objects of class k (k<=n), is an unordered subset composed of k objects, each can be chosen once
Cn,k=Dn,k/Pk=(n chooses k)=n!/(k!*(n-k)!)

In practice: When you need to chose all subset of a set of size k, whatever the order
(but once you choose the element i you cannot rechoose it --> simple)

Example: How many strings of 2 chars can be formed from the vowels (5)
C=5!/(2!*(5-2)!)=(5*4)/2=10 strings

Implementation (data structure):

```
//mark is not needed when we do not deal with repetitions since it forces one
possile ordering: precendece to the first in order of the subset
val=malloc(n*sizeof(int));
//the set
sol=malloc(k*sizeof(int));
//solution

Implementation (algorithm):
int comb(int *val, int *sol, int n, int k, int start, int count, int pos){
    int i;
    if (pos>=k){
        printf("Solution %d: ,count);
        for (i=0;i<k;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=start;i<n;i++){
        sol[pos]=val[i];
        count=comb(val,sol,n,k,i+1,count,pos+1); //recur on i(values) and
pos(shifting to the next spot in the solution)
    }
    return count;
}

Example:
(AEIOU)
AE
AI
AO
AU
EI
EO
EU
IO
IU
OU



---7-COMBINATIONS-WITH-REPETITIONS

Given one set, combinations with repetitions, of n distinct objects of class k
(k<=n), is an unordered subset composed of k objects, each can be chosen more
times
Cn,k=(n+k-1 chooses k)=(n+k-1 chooses n-1)=(n+k-1)!/(k!*(n-1)!)

In practice: When you need to chose all subset of a set of size k, whatever the
order
(and you can choose the elements more times)

Example: How many compositions of values may appear when simultanously casting
two dices?
C'=(6+2-1)!/(2!*(6-1)!)=(7*6)/2=21 compositions

Implementation (data structure):
//mark is not needed when we do not deal with repetitions since it forces one
possile ordering: precendece to the first in order of the subset
val=malloc(n*sizeof(int));
//the set
sol=malloc(k*sizeof(int));
//solution

Implementation (algorithm):
//as simple combination but i is not incremented when recurring, so the same
object is reconsidered
int comb_rep(int *val, int *sol, int n, int k, int start, int count, int pos){
    int i;
    if (pos>=k){
```

```
        printf("Solution %d: ,count);
        for (i=0;i<k;i++)
            printf("%d ",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=start;i<n;i++){
        sol[pos]=val[i];
        count=comb_rep(val,sol,n,k,i,count,pos+1); //recur on pos only (i is not
incremented)
    }
    return count;
}
```

Example:

```
11
12
13
14
15
16
22
23
24
25
26
33
34
35
36
44
45
46
55
56
66
```

---8-EXERCISES

a. Anagrams

```
#include <stdio.h>
#define N 256
#define MAX 20

int freq(char *string, int *freq);
int anags(char *anag, int *freq, int len, int pos, int count);

int main(void){
    int freqs[N], len;
    char word[MAX], anag[MAX];
    scanf("%s",word);
    len=freq(word,freqs);
    anag[len]='\0';
    return anags(anag,freqs,len,0,0);
}

int freq(char *string, int *freqs){
    int i;
    for (i=0;i<N;i++)
        freqs[i]=0;
    for (i=0;string[i]!='\0';i++)
            freqs[string[i]]++;
    return i;
}

int anags(char *anag, int *freqs, int len, int pos, int count){
```

```
    int i;
    if (pos>=len){
        printf("%s\n",anag);
        return count+1;
    }
    for (i=0;i<N;i++){
        if (freqs[i]>0){
            freqs[i]--;
            anag[pos]=i;
            count=anags(anag,freqs,len,pos+1,count);
            freqs[i]++;
        }
    }
    return count;
}

//this solution uses permutations with repetitions, based on ASCII, and the
order of the solutions is in order by ASCII
```

b. Powerset

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

int power(int *arr, int n);
int comb(int *arr, int *sol, int k, int n, int start, int pos, int count);

int main(void){
    int n, *arr, i, count;
    scanf("%d",&n);
    arr=malloc(n*sizeof(int));
    if (arr==NULL)
        return 1;
    for (i=0;i<n;i++)
        scanf("%d",&arr[i]);
    count=power(arr,n);
    free(arr);
    return count;
}

int power(int *arr, int n){
    int i,count=0, *sol, *mark;
    sol=malloc(n*sizeof(int));
    if (sol==NULL)
        return 1;
    for (i=0;i<=n;i++)
        count+=comb(arr,sol,i,n,0,0,0);
    free(sol);
    return count;
}

int comb(int *arr, int *sol, int k, int n, int start, int pos, int count){
    int i;
    if (pos>=k){
        for (i=0;i<k;i++)
            printf("%d",sol[i]);
        printf("\n");
        return count+1;
    }
    for (i=start;i<n;i++){
        sol[pos]=arr[i];
        count=comb(arr,sol,k,n,i+1,pos+1,count);
    }
    return count;
}


//unordered, k=1ton, no rep --> simple combinations with k from 1 to n
```