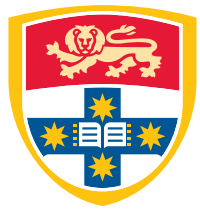
Introduction to Programming (Adv)

School of Computer Science, University of Sydney



Copyright Warning

Lecture 3: Recursion in C

Understanding memory

Recursion

Solve the same problem on a different set of data, or using different parameters

Solving in this way leads to a global solution

Recursion problems

Factorial. n! = n*(n-1)!

```
1  int fac(int n) {
2     if (n <= 1)
3        return 1;
4     return n*fac(n-1);
5  }
```

```
1  int fac(int n) {
2     if (n <= 1)
3        return 1;
4     return fac(n-1) * n;
5  }
```

Recursion

The problem

$$f(x,y,z) = \begin{cases} x-y, & \text{if } x+y+z > 10 \\ f(x+y, y-1, z-1), & \text{if } x+y < 3 \\ f(x/2, y, z), & \text{otherwise} \end{cases}$$

Consider as integers

```
1  int f(int x, int y, int z) {
2      if (x + y + z > 10)
3          return x - y;
4
5      if (x + y < 3)
6          return f(x+y, y-1, z-1)
7
8      return f(x/2, y, z)
9  }
```

Produce all orderings

Given a string of characters, produce all permutations of the characters.

e.g. ABCD

ABCD	BACD	CBAD	DBCA
ABDC	BADC	CBDA	DBAC
ACBD	BCAD	CABD	DCBA
ACDB	BCDA	CADB	DCAB
ADCB	BDCA	CDAB	DACB
ADBC	BDAC	CDBA	DABC

Produce all orderings (cont.)

Recursive thinking: to solve the problem of permuting n characters, we need to solve the problem of permuting $n-1$ characters where the first character is swapped with another.

A as first letter and all its combinations

B as first letter and all its combinations

C as first letter and all its combinations

D as first letter and all its combinations

```
A{.remainder.} = AB{.remainder.} = ABC{.remainder.} = ABCD
A{.remainder.} = AB{.remainder.} = ABD{.remainder.} = ABDC
A{.remainder.} = AC{.remainder.} = ACB{.remainder.} = ACBD
...
```

Produce all orderings (cont.)

General principle:

```
swap one character with the first
produce all permutations with that leading character.
print each permutation
swap first character with the original (swap back!)
```

when do we stop recursion?
no more permutations, i.e. size of remaining problem is 1

The base case is where we get the permutation

Produce all orderings (cont.)

Recursion may require a wrapper function

Setting up and passing along parameters for recursion

Our interface may be:

```
void permute(const char *string);
```

This does not help us pass along parameters, or return values

Produce all orderings (cont.)

wrapper function

```
1  // wrapper function. public facing. Setups our recursion
2  void permute(const char *s)
3  {
4      char data[2048];
5      size_t len = strlen(s);
6      strncpy(data, s, len);
7      data[len] = '\0';
8      permute_r(data, 0, len);
9      printf("\n");
10 }
11 int main() {
12     permute("ABCD");
13     return 0;
14 }
```

Produce all orderings (cont.)

```
1  void permute_r(char *s, int start, size_t len) {
2      if (start >= len) {
3          printf("%s\n", s);
4          return;
5      }
6      char first = s[start];
7      for (int i = start; i < len; i++) {
8
9          // swap first letter with other letter
10         char tmp = s[i];
11         s[i] = first;
12         s[start] = tmp;
13
14         // solve the sub problem of
15         // permuting remaining string
16         permute_r(s, start+1, len);
17
18         // swap back
19         s[i] = tmp;
20         s[start] = first;
21     }
22 }
```

Produce all orderings (cont.)

How else could we achieve this?

Here is a version with pointer arithmetic

```
1  void permute(const char *s)
2  {
3      char data[2048];
4      size_t len = strlen(s);
5      strncpy(data, s, len);
6      data[len] = '\0';
7      permute_r(data, data);
8      printf("\n");
9  }
```

Produce all orderings (cont.)

```
1  void permute_r(char *s, char *sub) {
2      size_t len = strlen(sub);
3      if (len <= 1) {
4          printf("%s\n", s);
5          return;
6      }
7      char first = sub[0];
8      for (int i = 0; i < len; i++) {
9          // swap first letter with other letter
10         char tmp = sub[i];
11         sub[i] = first;
12         sub[0] = tmp;
13
14         // solve the sub problem of
15         // permuting remaining string
16         permute_r(s, sub+1);
17
18         // swap back
19         sub[i] = tmp;
20         sub[0] = first;
21     }
22 }
```

Produce all orderings (cont.)

Tweaks and changes

Adjust the code to report the permutation number with the string ONLY using parameters of the recursive function.

0: ABCD	6: BACD	12: CBAD	18: DBCA
1: ABDC	7: BADC	13: CBDA	19: DBAC
2: ACBD	8: BCAD	14: CABD	20: DCBA
3: ACDB	9: BCDA	15: CADB	21: DCAB
4: ADCB	10: BDCA	16: CDAB	22: DACB
5: ADBC	11: BDAC	17: CDBA	23: DABC

Try to improve upon this to handle combinations. To make life a little easier, consider the input is already sorted: e.g. AABB

Summary

When we identify recursion we need to plan how we will operate on the data.

Wrapper functions need to be setup to support recursion, along with any assumptions

Passing parameters demands an understanding in shared vs copied memory