CSCI 2132
Software Development

Lecture 19:

Generating Permutations

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Mergesort implementation and discussion
- Mergesort complexity calculation
- Quicksort vs Mergesort
- Example: Generating permutations

Planning Permute Algorithm

- The algorithm will be recursive
- Algorithm will keep first k elements fixed and permute the rest
- Algorithm parameters: `Permute(A, k, n)`
- Initial call to the algorithm:
  `Permute(A, 0, n)`
- Base case: `k == n-1`

Pseudocode for Generating Permutations

```
Permute(A, k, n)   /* version 1 */
INPUT: A - the array containing numbers 1..n in some
           order
       n - the length of array A
       k - (0<=k<=n) indicating k elements A[0]..A[k-1]
           that will be fixed
OUTPUT: prints all permutations of A that keep first k
        elements fixed, and at the end leaves the order
        of elements of A the same as initially found
1: IF k == n-1 THEN print A
2: ELSE
3:   Permute(A, k+1, n)
4:   FOR i = k+1 TO n-1 DO
5:     swap A[k] with A[i]
6:     Permute(A, k+1, n)
7:     swap A[k] with A[i]    /* swap back */
```

Pseudocode for Generating Permutations

```
Permute(A, k, n)   /* version 2 */
INPUT: A - the array containing numbers 1..n in some
           order
       n - the length of array A
       k - (0<=k<=n) indicating k elements A[0]..A[k-1]
           that will be fixed
OUTPUT: prints all permutations of A that keep first k
        elements fixed, and at the end leaves the order
        of elements of A the same as initially found
1: IF k == n-1 THEN print A
2: ELSE
3:   FOR i = k TO n-1 DO
4:     swap A[k] with A[i]
5:     Permute(A, k+1, n)
6:     swap A[k] with A[i]    /* swap back */
```

- Let us look at C code: `~prof2132/public/permute.c-blanks`

```
/* Program: permute.c
   Purpose: prints all permutations of numbers 1..LEN
*/
#include <stdio.h>
#define LEN 4

void swap(int array[], int i, int j);
void permute(int array[], int k, int n);

int main() {
  int array[LEN], i;

  for (i = 0; i < LEN; i++)
    array[i] = i + 1;

  permute(array, 0, LEN);
}
```

```
/* Function permute prints all permutations of the
   array 'array' of length n, such that the first k
   elements are not permuted (they stay fixed).

 Parameters:
   array - the array of elements to be permuted
   n - the length of array 'array'
   k - the first k elements of array are note permuted
*/
void permute(int array[], int k, int n) {
  int i;

  if ( _______________ ) {
    for (i = 0; i < n; i++)
      printf("%d ", array[i]);
    printf("\n");
  }
```

```
  else {
    for (i = k; i < n; i++) {
      swap(array, k, i);
      permute(array, _______ , _________ );
      swap(array, k, i);
    }
  }
}

/* Function swaps swaps the elements array[i] and
   array[j] */
void swap(int array[], int i, int j) {
  int temp = array[i];
  array[i] = array[j];
  array[j] = temp;
}
```

Multidimensional Arrays as Arguments

- Multidimensional arrays can be passed as function arguments
- Compiler must know all dimensions, except optionally the first
- C89 and earlier: Dimensions had to be constant
  - Example: `int f(a[N][M]) {...}` if `N` and `M` are constants
- C99 and later allows non-constant dimensions expressed using parameters and constants; example:

  ```
  int f(int n, int a[][n]);
  ```
  or
  ```
  int f(int n, int a[][*]);
  ```
- Important than `n` parameter comes before `a` parameter
- Definition examples:

  ```
  int f(int n, int a[][n]) {...
  ```
  or
  ```
  int f(int n, int a[][n+1]) {...
  ```
- Not valid:

  ```
  int f(int a[][n], int n) {...
  ```

Code Example (Compiler Error)

```
#include <stdio.h>

/* ERROR: Compiler will not allow this: */
void fun_print(int n, int a[][]);

int main() {
  int n;
  printf("Enter matrix dim: ");
  scanf("%d", &n);
  int a[n][n];
  for (int i=0; i<n; ++i)
    for (int j=0; j<n; ++j)
      a[i][j] = i*n+j;
  fun_print(n, a);
  return 0;
}
```

```
void fun_print(int n, int a[][]) {
  for (int i=0; i<n; ++i) {
    for (int j=0; j<n; ++j)
      printf(" %2d", a[i][j]);
    printf("\n");
  }
}
```

Code Example (Correct Version, C99)

```
#include <stdio.h>
void fun_print(int n, int a[][n]);
// Important: n must come before a
// E.g. not valid: void fun_part(int a[][n], int n);

int main() {
  int n;
  printf("Enter matrix dim: ");
  scanf("%d", &n);
  int a[n][n];
  for (int i=0; i<n; ++i)
    for (int j=0; j<n; ++j)
      a[i][j] = i*n+j;
  fun_print(n, a);
  return 0;
}
```

```
void fun_print(int n, int a[][n]) {
  for (int i=0; i<n; ++i) {
    for (int j=0; j<n; ++j)
      printf(" %2d", a[i][j]);
    printf("\n");
  }
}
```

Program Organization

- Reading: Ch10 King, Program Organization
- **Local Variables:**
- Defined inside the body of a function
- Stored in the stack part of memory (call stack)
- Allocated automatically in memory when stack frame is created
  - this is called: *automatic storage duration*
- Have block scope
- Function parameters are similar but always initialized
- Using keyword `static` we can use local static variables, which are stored in *data*