Recursion (cont.)

Instructor: Dr. Sunho Lim (Ph.D., Assistant Professor)

Lecture 07

sunho.lim@ttu.edu

Adapted partially from Data Structures and Algorithms in C++, Adam Drozdek, 4th Edition, Cengage Learning; and Algorithms and Data Structures, Douglas Wilhelm Harder, Mmath

Tail Recursion

- The nature of a recursive definition
  - the function contains a reference to itself
  - This reference can take on a number of different forms
- Starting with the simplest, ***tail recursion***
  - a **single** recursive call occurs **at the end of the function**
  - **no other statements** follow the recursive call
  - **no other recursive calls** prior to the call at the end of the function

Tail Recursion (cont.)

- e.g., a tail recursive function:

```
void tail(int i) {
   if (i > 0) {
      cout << i << '';
      tail(i-1);
   }
}
```

- Tail recursion,
  - a loop
  - can be replaced by an **iterative algorithm** to accomplish the same task

Tail Recursion (cont.)

- e.g., an iterative form of the function:

```
void iterativeEquivalentOfTail(int i) {
   for ( ; i > 0; i--)
      cout << i << '';
}
```


Nontail Recursion

- e.g., an another type of recursion:

```
/* 200 */     void reverse() {
                  char ch;
/* 201 */         cin.get(ch);
/* 202 */         if (ch != '\n') {
/* 203 */             reverse();
/* 204 */             cout.put(ch);
                  }
              }
```

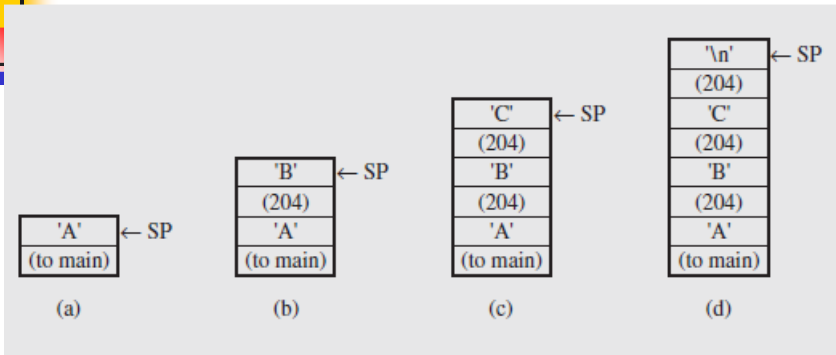
  - the recursive call precedes other code in the function
    - **nontail recursion**
  - display a line of input in **reverse order**
  - assuming the input, "ABC"

Nontail Recursion (cont.)

```
/* 200 */     void reverse() {
                  char ch;
/* 201 */         cin.get(ch);
/* 202 */         if (ch != '\n') {
/* 203 */             reverse();
/* 204 */             cout.put(ch);
                  }
              }
```

- The first time reverse() is called…
  - an **activation record** is created to store the **local variable** ch and the **return address** of the call in main()

Nontail Recursion (cont.)



- When the end of line character is read,
  - the snapshot of stack appeared
  - **terminate** the current call
  - **popping** the last activation record off the **stack**
  - **resuming** the previous call

Nontail Recursion (cont.)

- Consider a non-recursive version of the same algorithm:

```
void simpleIterativeReverse() {
   char stack[80];
   register int top = 0;
   cin.getline(stack,80);
   for(top = strlen(stack)-1; top >= 0;
 cout.put(stack[top--]));
}
```

  - utilize functions from the **standard C++ library** to handle the input and string reversal

Nontail Recursion (cont.)

- Make the processing more explicit,

```
void iterativeReverse() {
    char stack[80];
    register int top = 0;
    cin.get(stack[top]);
    while(stack[top]!='\n')
        cin.get(stack[++top]);
    for (top -= 1; top >= 0;
          cout.put(stack[top--]));
}
```


Nontail Recursion (cont.)

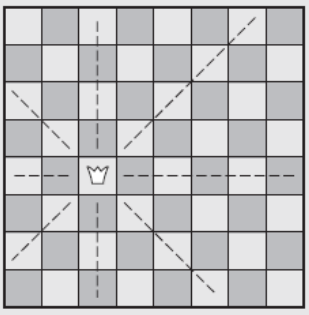
- Consider these iterative versions:
  - when implementing nontail recursion **iteratively**,
    - the **stack** must be explicitly implemented and handled
  - the **clarity of the algorithm** are sacrificed as a consequence of the conversion
  - unless compelling reasons are evident during the algorithm design,
    - use the **recursive versions**

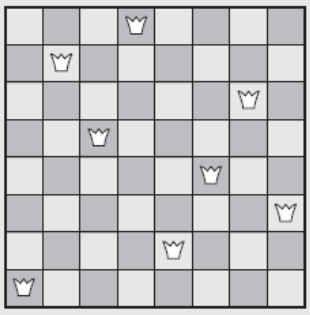
Backtracking

- ***Backtracking***
  - an approach to problem solving that uses a systematic search among possible pathways to a solution
- As each path is examined, if it is determined the pathway isn't viable,
  - it is discarded and the algorithm returns to the **prior branch**
  - **a different path** can be explored
- The algorithm must be able to return to the previous position,
  - ensure **all pathways** are examined
- Potential applications,
  - artificial intelligence, compiling, and optimization problems
  - e.g., ***The Eight Queens Problem*** – *no two queens share the same row, column, or diagonal*

Backtracking (cont.)

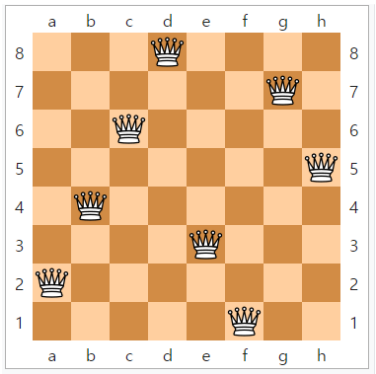
- Try to place eight queens on a chessboard in such a way
  - no two queens attack each other

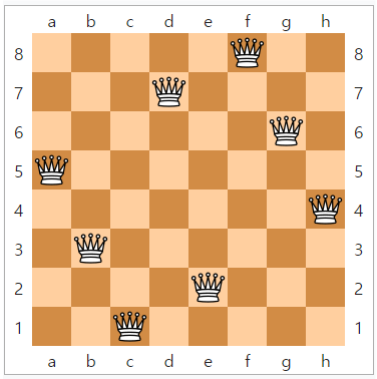




Backtracking (cont.)

- Try to place eight queens on a chessboard in such a way
  - no two queens attack each other (cont.)



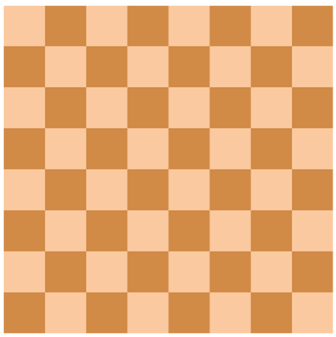


Backtracking (cont.)

- Place one queen at a time,
  - trying to make sure that the queens do not check each other
- If at any point a queen cannot be successfully placed,
  - backtrack to the placement of the previous queen
  - the next queen is tried again
- If no successful arrangement is found,
  - backtracks further
  - adjust the previous queen's predecessor, etc.

Backtracking (cont.)

```
putQueen(row)
   for every position col on the same row
     if position col is available
        place the next queen in position col;
        if (row < 8)
          putQueen(row+1);
        else success;
        remove the queen from position col;
```



- This algorithm will find all solutions, although some are symmetrical