Recursive-Descent Parsing

- Top-down parsing strategy
- One function/procedure for each nonterminal
- Functions call each other recursively, based on the grammar
- Recursion stack handles the tasks of LL(1) parser stack
- LL(1) conditions to be satisfied for the grammar
- Can be automatically generated from the grammar
- Hand-coding is also easy
- Error recovery is superior

An Example

Grammar: $S' \rightarrow S\$,\ S \rightarrow aAS \mid c,\ A \rightarrow ba \mid SB,\ B \rightarrow bA \mid S$

```
/* function for nonterminal S' */
void main(){/* S' --> S$ */
   fS(); if (token == eof) accept();
         else error();
}
/* function for nonterminal S */
void fS(){/* S --> aAS | c */
   switch token {
     case a : get_token(); fA(); fS();
              break;
     case c : get_token(); break;
     others : error();
   }
}
```

An Example (contd.)

```
void fA(){/* A --> ba | SB */
    switch token {
      case b : get_token();
               if (token == a) get_token();
               else error(); break;
      case a,c : fS(); fB(); break;
      others : error();
    }
}
void fB(){/* B --> bA | S */
    switch token {
      case b : get_token(); fA(); break;
      case a,c : fS(); break;
      others : error();
    }
}
```