```
Tower of Hanoi
       start
             finish temp
3
              В
 void MoveTower(int n, char start, char finish, char temp) {
         if (n == 1) {
             MoveSingleDisk(start, finish);
         } else {
            ->MoveTower(n - 1, start, temp, finish);
MoveSingleDisk(start, finish);
             MoveTower(n - 1, temp, finish, start);
} }
                     start finish temp
              n
              2
                     Α
                            С
                                   В
               void MoveTower(int n, char start, char finish, char temp) {
                       if (n == 1) {
                           MoveSingleDisk(start, finish);
                       } else {
                           ->MoveTower(n - 1, start, temp, finish);
                           MoveSingleDisk(start, finish);
                           MoveTower(n - 1, temp, finish, start);
              } }
                                   start finish temp
                            n
                                   Α
                                          В
                             void MoveTower(int n, char start, char finish, char temp) {
                                     if (n == 1) {
                                                                                                               1
                                         ->MoveSingleDisk(start, finish);
                                     } else {
                                         MoveTower(n - 1, start, temp, finish);
                                         MoveSingleDisk(start, finish);
                                         MoveTower(n - 1, temp, finish, start);
                            } }
                     start finish temp
              2
                     Α
                            С
               void MoveTower(int n, char start, char finish, char temp) {
                       if (n == 1) {
                                                                                                               2
                           MoveSingleDisk(start, finish);
                       } else {
                           MoveTower(n - 1, start, temp, finish);
                           ->MoveSingleDisk(start, finish);
                           MoveTower(n - 1, temp, finish, start);
              } }
                     start finish temp
              n
              2
                     Α
                            С
                                   В
               void MoveTower(int n, char start, char finish, char temp) {
                       if (n == 1) {
                           MoveSingleDisk(start, finish);
                       } else {
                           MoveTower(n - 1, start, temp, finish);
                           MoveSingleDisk(start, finish);
                           ->MoveTower(n - 1, temp, finish, start);
              } }
                                   start finish temp
                                   В
                                          С
                                                 Α
                            1
                             void MoveTower(int n, char start, char finish, char temp) {
                                     if (n == 1) {
                                         ->MoveSingleDisk(start, finish);
                                     } else {
                                         MoveTower(n - 1, start, temp, finish);
                                         MoveSingleDisk(start, finish);
                                         MoveTower(n - 1, temp, finish, start);
                            } }
                     start finish temp
              n
              2
               void MoveTower(int n, char start, char finish, char temp) {
                       if (n == 1) {
                           MoveSingleDisk(start, finish);
                       } else {
                           MoveTower(n - 1, start, temp, finish);
                           MoveSingleDisk(start, finish);
                           MoveTower(n - 1, temp, finish, start);
              ->} }
       start finish temp
n
3
 void MoveTower(int n, char start, char finish, char
temp) {
                                                                                                               4
         if (n == 1) {
             MoveSingleDisk(start, finish);
         } else {
             MoveTower(n - 1, start, temp, finish);
             ->MoveSingleDisk(start, finish);
             MoveTower(n - 1, temp, finish, start);
} }
       start finish temp
3
 void MoveTower(int n, char start, char finish, char
temp) {
         if (n == 1) {
            MoveSingleDisk(start, finish);
         } else {
             MoveTower(n - 1, start, temp, finish);
             MoveSingleDisk(start, finish);
             ->MoveTower(n - 1, temp, finish, start);
} }
                           finish temp
                     start
                     С
               void MoveTower(int n, char start, char finish, char temp) {
                        if (n == 1) {
                             MoveSingleDisk(start, finish);
                        } else {
                             ->MoveTower(n - 1, start, temp, finish);
                             MoveSingleDisk(start, finish);
                            MoveTower(n - 1, temp, finish, start);
              } }
                                          finish temp
                                   start
                                   С
                                                 В
                             void MoveTower(int n, char start, char finish, char temp) {
                                      if (n == 1) {
                                                                                                               5
                                           ->MoveSingleDisk(start, finish);
                                       } else {
                                           MoveTower(n - 1, start, temp, finish);
                                           MoveSingleDisk(start, finish);
                                           MoveTower(n - 1, temp, finish, start);
                            } }
                     start finish temp
               void MoveTower(int n, char start, char finish, char temp) {
                        if (n == 1) {
                            MoveSingleDisk(start, finish);
                        } else {
                            MoveTower(n - 1, start, temp, finish);
                             ->MoveSingleDisk(start, finish);
                                                                                                               6
                             MoveTower(n - 1, temp, finish, start);
              } }
                     start finish temp
                     С
                            В
                                Α
               void MoveTower(int n, char start, char finish, char temp) {
                        if (n == 1) {
                            MoveSingleDisk(start, finish);
                        } else {
                             MoveTower(n - 1, start, temp, finish);
                            MoveSingleDisk(start, finish);
                            -> MoveTower(n - 1, temp, finish, start);
              } }
                            n
                                   Start Finish Temp
                                          В
                             void MoveTower(int n, char start, char finish, char temp) {
                                      if (n == 1) {
                                                                                                               7
                                           ->MoveSingleDisk(start, finish);
                                      } else {
                                           MoveTower(n - 1, start, temp, finish);
                                           MoveSingleDisk(start, finish);
                                           MoveTower(n - 1, temp, finish, start);
                            } }
                     start finish temp
              2
               void MoveTower(int n, char start, char finish, char temp) {
                        if (n == 1) {
                            MoveSingleDisk(start, finish);
                        } else {
                            MoveTower(n - 1, start, temp, finish);
                             MoveSingleDisk(start, finish);
                            MoveTower(n - 1, temp, finish, start);
              ->} }
       Start Finish Temp
3
       Α
              В
                     С
void MoveTower(int n, char start, char finish, char temp) {
          if (n == 1) {
              MoveSingleDisk(start, finish);
          } else {
              MoveTower(n - 1, start, temp, finish);
              MoveSingleDisk(start, finish);
              MoveTower(n - 1, temp, finish, start);
->} }
```

pr 3	refix rest i ABC 0
RecursivePermute if (rest == print pr } else { for (int	<pre>(string prefix, string rest) { "") { efix;  i = 0; i &lt; n; i++) {</pre>
stri stri	ng newPrefix = prefix + rest[i]; ng newRest = rest.substr(0, i) + rest.substr(i+1); cursivePermute(newPrefix, newRest);
n	2 A BC 0  pid RecursivePermute(string prefix, string rest) {
	<pre>if (rest == "") {     print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);</pre>
}	->RecursivePermute(newPrefix, newRest);  n prefix rest i
	toid RecursivePermute(string prefix, string rest) {  if (rest == "") {  print prefix;  }
	<pre>} else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         -&gt;RecursivePermute(newPrefix, newRest); } }</pre>
	n prefix rest i 0 ABC 0 ABC
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {         -&gt;print prefix; } else {         for (int i = 0; i &lt; n; i++) {             string newPrefix = prefix + rest[i]; }</pre>
	<pre>string newRest = rest.substr(0, i) + rest.substr(i+1);</pre>
	n prefix rest i  1 AB C 1  void RecursivePermute(string prefix, string rest) {   if (rest == "") {
	<pre>print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         RecursivePermute(newPrefix, newRest);</pre>
n	->; ; }
vo	<pre>poid RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {</pre>
}	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1); -&gt; RecursivePermute(newPrefix, newRest);</pre>
	n         prefix         rest         i           1         AC         B         0           void RecursivePermute(string prefix, string rest) {
	<pre>if (rest == "") {     print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);</pre>
	->RecursivePermute(newPrefix, newRest); }  n prefix rest i
	<pre>1 ACB  void RecursivePermute(string prefix, string rest) {     if (rest == "") {         -&gt; print prefix;     } }</pre> ACB
	<pre>} else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         RecursivePermute(newPrefix, newRest); } }</pre>
	n         prefix         rest         i           1 AC         B         1
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {             string newPrefix = prefix + rest[i];         } }</pre>
	<pre>string newRest = rest.substr(0, i) + rest.substr(i+1);</pre>
n ve	2 A BC 2  pid RecursivePermute(string prefix, string rest) {    if (rest == "") {
	<pre>print prefix; } else {   for (int i = 0; i &lt; n; i++) {     string newPrefix = prefix + rest[i];     string newRest = rest.substr(0, i) + rest.substr(i+1);     RecursivePermute(newPrefix, newRest);</pre>
}	refix rest i ABC 1
RecursivePermute(     if (rest ==         print pr     } else {	<pre>string prefix, string rest) { "") { efix;</pre>
stri stri	<pre>i = 0; i &lt; n; i++) { ng newPrefix = prefix + rest[i]; ng newRest = rest.substr(0, i) + rest.substr(i+1); cursivePermute(newPrefix, newRest);</pre>
n ve	2 B AC 0  pid RecursivePermute(string prefix, string rest) {
	<pre>if (rest == "") {     print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);</pre>
}	-> RecursivePermute(newPrefix, newRest); ->
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {</pre>
	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1);  -&gt;RecursivePermute(newPrefix, newRest);  -&gt;} } }</pre>
	n prefix rest i  0 BAC 0  void RecursivePermute(string prefix, string rest) {     if (rest == "") {
	<pre>-&gt; print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         RecursivePermute(newPrefix, newRest); }</pre>
	n     prefix     rest     i       1 BA     C     1
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {             string newPrefix = prefix + rest[i];         } }</pre>
n	<pre>string newRest = rest.substr(0, i) + rest.substr(i+1); RecursivePermute(newPrefix, newRest);  -&gt;} } prefix rest i</pre>
VC	2 B  oid RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {
-	<pre>for (int i = 0; i &lt; n; i++) {</pre>
	n         prefix         rest         i           1         BC         A         0           void RecursivePermute(string prefix, string rest) {         Void RecursivePermute(string prefix, string rest) {
	<pre>if (rest == "") {     print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);</pre>
	->RecursivePermute(newPrefix, newRest); }
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {         -&gt; print prefix;     } else {         for (int i = 0; i &lt; n; i++) {</pre>
	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1); RecursivePermute(newPrefix, newRest); } } </pre>
	n prefix rest i  BC A 1  void RecursivePermute(string prefix, string rest) {   if (rest == "") {
	<pre>print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         RecursivePermute(newPrefix, newRest);</pre>
n	->} } prefix rest i  2 B AC 2
VC	<pre>pid RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {</pre>
}	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1); RecursivePermute(newPrefix, newRest);  } }</pre>
3	
} else { for (int stri stri	<pre>efix;  i = 0; i &lt; n; i++) {     ng newPrefix = prefix + rest[i];     ng newRest = rest.substr(0, i) + rest.substr(i+1);     cursivePermute(newPrefix, newRest);</pre>
n	2 C AB 0
vo	<pre>pid RecursivePermute(string prefix, string rest) {     if (rest == "") {         -&gt;print prefix;     } else {         for (int i = 0; i &lt; n; i++) {             string newPrefix = prefix + rest[i];         } }</pre>
}	<pre>string newRest = rest.substr(0, i) + rest.substr(i+1); -&gt; RecursivePermute(newPrefix, newRest); }</pre>
	n prefix rest i  1 CA B 0  void RecursivePermute(string prefix, string rest) {    if (rest == "") {
	<pre>-&gt;print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         -&gt; RecursivePermute(newPrefix, newRest); } </pre>
	} } } }        Image: Control of the contr
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {</pre>
	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1); RecursivePermute(newPrefix, newRest); } } }</pre>
	n prefix rest i  1 CA B 1  void RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix:
	<pre>print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         RecursivePermute(newPrefix, newRest);</pre>
n	->} }       prefix     rest     i       2 C     AB     1
V	<pre>pid RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {             string newPrefix = prefix + rest[i];         } }</pre>
}	<pre>string newRest = rest.substr(0, i) + rest.substr(i+1);    &gt;RecursivePermute(newPrefix, newRest); }</pre>
	toid RecursivePermute(string prefix, string rest) {  if (rest == "") {  print prefix;  A 0
	<pre>print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         -&gt; RecursivePermute(newPrefix, newRest); } }</pre>
	} } } }       n     prefix     rest     i       0     CBA     0
	<pre>void RecursivePermute(string prefix, string rest) {     if (rest == "") {         -&gt;print prefix;     } else {         for (int i = 0; i &lt; n; i++) {             string newPrefix = prefix + rest[i];         }</pre>
	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1); RecursivePermute(newPrefix, newRest); } }  n</pre> prefix rest i
	1 CB A 1  void RecursivePermute(string prefix, string rest) {    if (rest == "") {
	<pre>print prefix; } else {     for (int i = 0; i &lt; n; i++) {         string newPrefix = prefix + rest[i];         string newRest = rest.substr(0, i) + rest.substr(i+1);         RecursivePermute(newPrefix, newRest);</pre>
n	->} } }
vo	<pre>pid RecursivePermute(string prefix, string rest) {     if (rest == "") {         print prefix;     } else {         for (int i = 0; i &lt; n; i++) {</pre>
}	<pre>string newPrefix = prefix + rest[i]; string newRest = rest.substr(0, i) + rest.substr(i+1); RecursivePermute(newPrefix, newRest); &gt;} }</pre>
nr	refix rest i ABC 3 string prefix, string rest) {
RecursivePermute( if (rest ==	) { efix:
RecursivePermute( if (rest ==     print pr } else {     for (int     stri     stri	<pre> i = 0; i &lt; n; i++) {  ng newPrefix = prefix + rest[i];  ng newRest = rest.substr(0, i) + rest.substr(i+1);  rsivePermute(newPrefix, newRest); </pre>