Lecture – Statements

Expression statement

- Syntax
  exp;

Null statement

- Syntax
  ;
- Example

```
if (n<0) n=-n;
if (n>=0) ; else n=-n;

for (;;);
while(1);
while(true);
```

(the three loops above) infinite loop

Block statement

- Syntax
  { *stmt*/ *decl* ... }
- A block statement delimits the scope of variables declared in it.

```
{
    ................        ← a is invisible here
    int a = ........
    ................        ← a is visible here
}
```

Example

```
{                           {
    int a=2;  ?                 const int a=7;
    { int a=a; }                { int a[a]; }  ?
}                           }
```

Conditional statements

- Syntax
  if (*exp***)** *stmt* [**else** *stmt*]
  switch (*exp***)** *stmt*

- Example

```
if (n==0) s0;
else if (n==1) s1;
else if (n==2 ||n==3) s23;
else s4;
```

  is equivalent to

```
switch (n) {
case 0: s0;
        break;
case 1: s1;
        break;
case 2: case 3: s23;
        break;
default:s4;
}
```

```
switch(n)
{
    ⋮
case 1:
    ⋮
    ⋮
    ⋮
    ⋮
}
```

let **n=1**

- Notes on switch statement
  1. The *stmt* is usually a block containing labeled statements:
     case *const_exp*: *stmt*
     default: *stmt*
  2. The *exp* and *const_exp* must be of integral type.
     The *const_exp* is evaluated during compilation.
  3. The **case**'s and (at most one) **default** can only appear inside **switch** statements; they may appear in any order.
  4. Four ways to leave a **switch** statement
     ① break ② return ③ goto ④ fall out of the stmt
  5. Variables cannot be initialized inside a **switch** statement unless they are inside a block statement or visible only for the last case.

- Example

  `switch (n) case 0:` *do-something*`;`
  `if (n==0)` *do-something*`;`

- Example

```
unsigned fib(unsigned n)
{
   switch(n) {
   case 0: case 1: return n;
   default:
      unsigned s=0,r=1;
      for (int i=1;i<n;i++) { r=r+s; s=r-s; }
      return r;
   }
}
```

```
unsigned fib(unsigned n)
{
   switch(n) {
   default:
      unsigned s=0,r=1;     // error
      for (int i=1;i<n;i++) { r=r+s; s=r-s; }
      return r;
   case 0: case 1: return n;
   }
}
```

The declaration is unfair, because other cases can see **`s`** and **`r`**, but have no chance to initialize them.

Solutions

1 Let **`default`** be the last case

2 **`default: unsigned s,r; s=0; r=0; ...`**

3
```
default: {
      unsigned s=0,r=1;
      for (int i=1;i<n;i++) { r=r+s; s=r-s; }
      return r; }
```

N.B. The labels within inner blocks are still visible in the outer **`switch`** block.

Iteration statements

- Syntax

  while (*exp***)** *stmt*
  do *stmt* **while (***exp***);**
  for (*exp*1; *exp*2; *exp*3**)** *stmt*

- Example – Coin change

```
int cc(int n)
{
   int count=0;
   for (int c50=0;c50<=n/50;c50++)
   for (int c10=0;c10<=(n-50*c50)/10;c10++)
   for (int c5=0;c5<=(n-50*c50-10*c10)/5;c5++)
   {
      int c1=n-50*c50-10*c10-5*c5;
      printf("%5d%5d%5d%5d\n",c1,c5,c10,c50);
      count++;
   }
   return count;
}
```

To avoid recomputing the upper bound of each loop, do this:

```
for (int c50=0,m50=n/50;c50<=m50;c50++)
for (int c10=0,n10=n-50*c50,m10=n10/10;c10<=m10;c10++)
for (int c5=0,n5=n10-10*c10,m5=n5/5;c5<=m5;c5++)
{
   int c1=n5-5*c5; ...
}
```

To avoid computing the money left by multiplication, do this:

```
for (int c50=0,n50=n,m50=n50/50;c50<=m50;c50++,n50-=50)
for (int c10=0,n10=n50,m10=n10/10;c10<=m10;c10++,n10-=10)
for (int c5=0,n5=n10,m5=n5/5;c5<=m5;c5++,n5-=5)
{
   int c1=n5; ...
}
```

- Example (Cont'd)

  Now that the money left is known, we may finally write

```
for (int c50=0,n50=n;n50>=0;c50++,n50-=50)
for (int c10=0,n10=n50;n10>=0;c10++,n10-=10)
for (int c5=0,n5=n10;n5>=0;c5++,n5-=5)
{
   int c1=n5; ...
}
```

Jump statements

- Syntax
  return [*exp*]**;**
  break;
  continue;
  goto *label***;**
  where *label* is the label of a labeled statement:
  *label***:** *stmt*
  The syntax of *label* is the same as that of identifiers.

- A **continue** statement causes the next iteration of the inner-most enclosing loop to begin.

- Example

```
void primegen(int n)
{
   for (int k=2;k<=n;k++) {
      if (!prime(k)) continue;
      printf("%d ",k);
   }
}
```

- Example (Cont'd)

```
void primegen(int n)
{
   int k=2;
loop:                                   // if + goto = loop
   if (k>n) goto exit;
   if (prime(k)) printf("%d ",k);
   k++;
   goto loop;
exit:;
}
```

Remarks

1 Labels are in a different name space.
2 The scope of a label covers the entire function.

```
void primegen(int n)
{
   int k=2;
n: if (k>n) goto k;
   if (!prime(k)) {
      primegen: k++; goto n;
   }
   printf("%d ",k);
   goto primegen;
k:;
}
```

Remarks

1 Gotos considered harmful.
2 Structured programming
  Programming without gotos or with reliable use of gotos.

Examples

Example 1 – Echo

Version A

```
void echo(void)
{
   int ch;
   while ((ch=getchar())!=EOF) putchar(ch);
}
```

Remarks

1 `int getchar(void);`
returns a value of type **int** so as to accommodate to **EOF**.

2 `int putchar(int);`
returns the character written, e.g.
`printf("%d",puchchar(256+'a'));` // **a97**

Alternatively, we may treat the eoln (end-of-line) character as a marker marking the end of data in a line, rather than as a datum by itself.

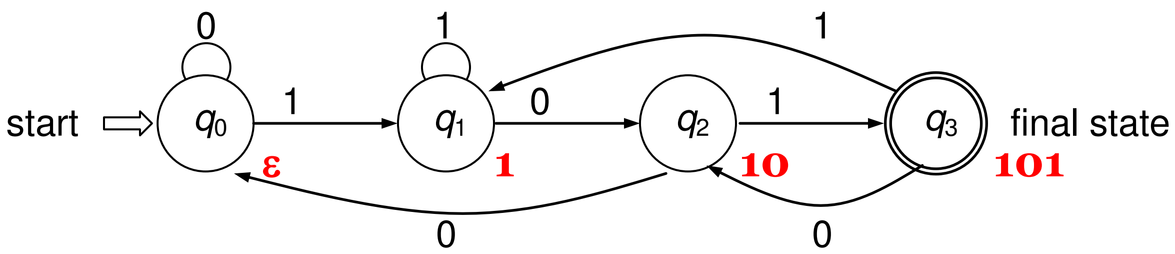
```
void echoline(void)
{
   int ch;
   while ((ch=getchar())!='\n') putchar(ch);
   printf("\n");
}

void echo(void)
{
   int ch;
   while ((ch=getchar())!=EOF) {
      ungetc(ch,stdin); echoline();
   }
}
```

Remarks: `getchar() = getc(stdin);`
`putchar(ch) = putc(ch,stdout)`

Example 2 – Deterministic Finite (state) Automaton

Problem : Determine if a binary string ends with 101

Transition diagram of a DFA



Let M be this DFA, then

$L(M)$
= the language accepted by M
= { x | x is a binary string such that M halts at state q_3 after reading }
= the set of all binary strings ending with 101

Formal languages and automata

A study of computability via computation models such as DFA, etc.
For example, the problem

Given an integer $n \geq 0$, is $n\%8 = 5$?

is computable, as the finite automation M solves it:

Let x be the binary representation of n, then

$n\%8 = 5 \Leftrightarrow x$ ends with 101 $\Leftrightarrow x \in L(M)$

Data representation

How to represent "states"?

1 As labels (Version 1)
```
q0: switch …;
```

2 As integers
```
const int q0=0,q1=1,q2=2,q3=3;
int q=q0;
```
Drawback: The states are integers, rather than new objects.
Cf.
```
#define q0 0
```
Drawback: Macros aren't subject to the scope rules.

Example 2 (Cont'd)

3 As enumerators (Versions 2 & 3)

```
enum state {q0,q1,q2,q3} q=q0;
```

Cf.

```
typedef int state;
```

Drawback: **state** isn't a new type.

4 As functions (See next lecture)

Version 1

```
void dfa(void)
{
q0: switch (getchar()) {
      case '0' : goto q0;
      case '1' : goto q1;
      case '\n': printf("Rejected\n"); return;
    }
q1: switch (getchar()) {
      case '0' : goto q2;
      case '1' : goto q1;
      case '\n': printf("Rejected\n"); return;
    }
q2: switch (getchar()) {
      case '0' : goto q0;
      case '1' : goto q3;
      case '\n': printf("Rejected\n"); return;
    }
q3: switch (getchar()) {
      case '0' : goto q2;
      case '1' : goto q1;
      case '\n': printf("Accepted\n"); return;
    }
}
```

N.B.
The statement labeled **q0** must be the 1st statement. The remaining three labeled statements may be permuted.

Example 2 (Cont'd)

```
int main(void)
{
   printf("Enter a binary string: ");
   int ch;
   while ((ch=getchar())!=EOF) {
      ungetc(ch,stdin);
      void dfa(void); dfa();
      printf("Enter a binary string: ");
   }
}
```

Version 2

```
void dfa(void)
{
   enum state {q0,q1,q2,q3} q=q0;  ①
   int ch;
   while ((ch=getchar())!='\n')
      switch (q) {
      case q0: q=ch=='0'? q0: q1; break;
      case q1: q=ch=='0'? q2: q1; break;
      case q2: q=ch=='0'? q0: q3; break;
      case q3: q=ch=='0'? q2: q1; break;
      }
   if (q==q3) printf("Accepted\n");
   else printf("Rejected\n");
}
①  enum state {q0,q1,q2,q3};
   enum state q=q0;
```

Version 3

	0	1
q_0	q_0	q_1
q_1	q_2	q_1
q_2	q_0	q_3
q_3	q_2	q_1

Transition table

Example 2 (Cont'd)

```
void dfa(void)
{
   enum state {q0,q1,q2,q3} q=q0;
   enum state trans[4][2]
                 ={{q0,q1},{q2,q1},{q0,q3},{q2,q1}};
   int ch;
   while ((ch=getchar())!='\n')
      q=trans[q][ch-'0'];
   if (q==q3) printf("Accepted\n");
   else printf("Rejected\n");;
}
```

On enumeration types

1 An enumeration type is a set of enumerators.
enum [*type*] **{***enumerator*, …**}** [*var*, …]**;**
Cf.
struct [*type*] **{***T x*; … **}** [*var*, …]**;**
union [*type*] **{***T x*; … **}** [*var*, …]**;**

2 Integral types are built-in enumeration types.

```
enum bool {false,true};
enum char {...,'a','b','c',...};
enum int  {INT_MIN,...,-1,0,1,...,INT_MAX};
```

3 Each enumerator is assigned an integer value.

```
enum color {red,green,blue};          // 0,1,2
enum color {red=2,green,blue=2};      // 2,3,2
```

4 In C,**sizeof(enum color)=sizeof(int)**
In C++, it is undefined.

5 In C, enumerators and integers may be mixed up.

```
enum color {red,green,blue};
enum weekend {sat,sun};
enum color c=red;       // 0
enum weekend d=sun;     // 1
int x=c+d;  // legal in C++               enum → int
c=x;        // in C++: c=(color)x;        int → enum
d=c;        // in C++: d=(weekend)x;      enum → enum
```

Example 2 (Cont'd)

6 In C++, when a user-defined type name is used, the keywords **enum**, **struct**, **union**, etc, may be absent, e.g.

```
state q;
```

In C, they must be present. However, they may be removed as follows:

```
typedef enum {qo,q1,q2,q3} state;
```

or

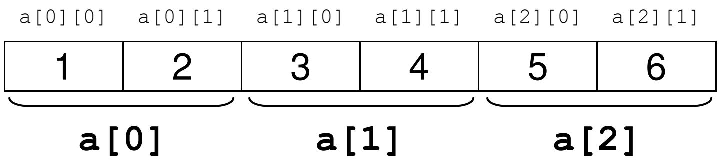
```
enum state {qo,q1,q2,q3};
typedef enum state state;
```

On array initialization

```
int a[3]={1,2,3};
int a[3]={1,2,3,4};       // error
int a[3]={1};             // ok, the trailing elements are 0.
int a[]={1,2,3};          // ok, array size=3
```

On two-dimensional arrays

In C/C++, a two-dimensional array is indeed a one-dimensional array whose elements themselves are one-dimensional arrays.



Two-dimensional arrays may be initialized as follows:

```
int a[3][2]={{1,2},{3,4},{5,6}};
int a[3][2]={1,2,3,4,5,6};
```

In either case, the 1st dimension may be absent, i.e. **int a[][2]**; but the 2nd dimension must be present, i.e. **int a[3][]** is illegal. (This will be explained later on.)

Note:

```
int a[3][2]={{1},{2},{3}};    // {{1,0},{2,0},{3,0}}
int a[3][2]={1,2,3};          // {{1,2},{3,0},{0,0}}
```

Example 3 – Sorting and searching

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

const int sz=20;                    // global
int a[sz];                          // global

void in(int n)                      // main → in  in → time
{                                   //   in: n; out: a    out: t
   time_t t;                   ①
   srand(time(&t));            ②   // time(&t); srand(t);
   for (int i=0;i<n;i++)           // srand(time(0));
      a[i]=rand()%100;         ③
}

void out(int n)                     // main → out
{                                   //      in: n, a
   for (int i=0;i<n;i++)
      printf("%d ",a[i]);
   printf("\n");
}

int main(void)                      // main → ?sort
{                                   //   in: n; inout: a
   int n=sz;                        // or any number ≤ sz
   in(n);
   printf("Before sorting ...\n"); out(n);
   ?sort(n);
   printf("After sorting ...\n"); out(n);
}
```

① **time_t** is an implementation-dependent arithmetic type representing times, say
typedef long time_t;

② **time_t time(time_t*);**

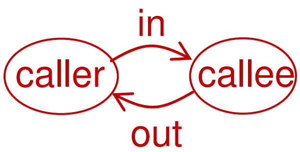
time(&t) usually returns the number of seconds elapsed since 1970/1/1 00:00:00; the value is also assigned to **t**.
time(0) only returns that value.

Example 3 (Cont'd)

② **`void srand(unsigned);`**
Set the seed to the given unsigned integer; the initial seed is 1.

③ **`int rand(void);`**
Generate an integer uniformly at random in the range
0 ≤ **`rand()`** ≤ **`RAND_MAX`**
32767 ≤ **`RAND_MAX`**

On communication between caller and callee



There are two ways for a caller and a callee to communicate with each other:

1 through global variables, when the target is fixed
In this case, the communication may be one- or two-way, e.g. the one- or two-way global array **`a`** in this example

2 through parameters, when the target is fixed or varied
In this case, the communication depends on parameter-passing methods:
2.1 Call-by-value is one-way
E.g. the one-way parameter **`n`** in this example
2.2 Call-by-reference may be one or two-way.
To be discussed later.

Q: Which is better when the target is fixed?
A: Global variables are better, as it takes time and space to pass and access parameters.

Example 3 (Cont'd)

Pseudo-random number generators (PRNGs)

A pseudo-random number generator generates arithmetically a sequence of numbers

$x_0\ x_1\ x_2\ \ \ldots\ \ \ x_n\ x_{n+1}\ \ \ldots$

Being generated by a *deterministic* algorithm, these numbers are not truly random – they only approximate randomness.
For example, periodicity is an inherent nonrandomness feature of a pseudo-random number generator.

Linear congruential generators (LCGs)

$x_{n+1} = (ax_n + b) \bmod c$ $\quad$ a, b, and c are suitably-chosen constants

Most common PRNGs implemented in standard libraries are LCGs. LCGs are best-known, easy-to-implement, and fast generators; but, they have severe defects, e.g. the lower order bits are much less random than the higher order bits.

VC++ implementation

```
#define RAND_MAX 32767
unsigned x=1;
void srand(unsigned s) { x=s; }
int rand(void)
{
   x=214013*x+2531011;        // 0 ≤ x < 2^32  (mod 2^32)
   return (x>>16)&0x7FFF;     // 0 ≤ rand() < 2^15
}
```

Observations

1. **srand** and **rand** communicates through the global variable **x**.
2. The computation makes use of modular arithmetic of unsigned integers.
3. The 16 less-random lower order bits are discarded.
   Note in particular the cycles:
   Last bit 0101…
   Last 2 bits 03210321....

Example 3 (Cont'd)

Concluding remarks

1 In case **`rand()`** is implemented as an LCG, do not rely on the lower order bits alone.

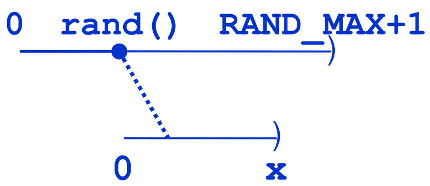
For example, how to express the result of casting a dice?

`1+rand()%6` // no

`1+randi(6)` // ok, $0 \le$ **`randi(6)`** < 6 (see below)

// PRNG for floating-point random numbers in the range $[0, x)$

```
double randd(double x)
{
   return rand()/(RAND_MAX+1.0)*x;
}
```



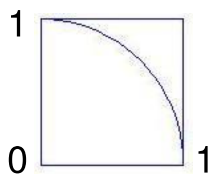
// PRNG for integral random numbers in the range $[0, n)$

```
int randi(int n)
{
   return randd(n);
}
```

2 If a large amount of random numbers is needed, use a better PRNG, e.g. Mersenne Twister.

Example – Monte-Carlo simulation

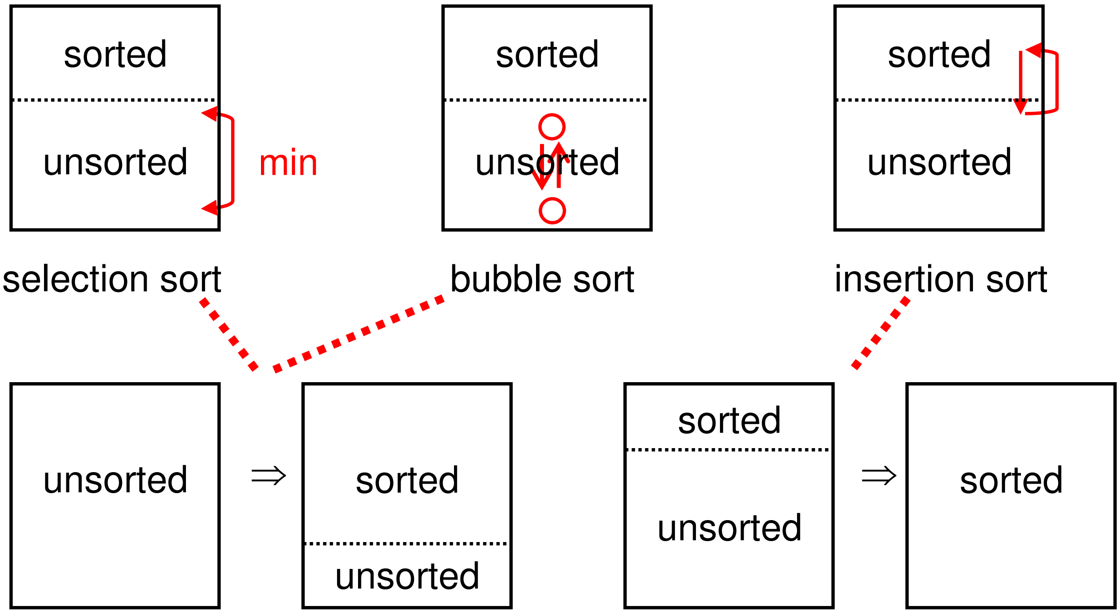
Estimation of π by shooting darts randomly at the figure below:



Let throws = # of darts thrown
hits = # of darts that hit the quadrant
Then,
hits / throws → π/4 as throws → ∞

Example 3 (Cont'd)

On sorting algorithms



Selection sort – $O(n^2)$

$i \to 0$	4		1		1		1		1
	3	$i \to 1$	3		2		2		2
	1	$\Rightarrow$	4	$\leftarrow j$ $\Rightarrow$	4	$\Rightarrow$	3	$\Rightarrow$	3
	5		5		5	$i \to n-2$	5		4
$n-1$	2		2		3		4		5

```
void ssort(int n)
{
    for (int i=0;i<n-1;i++) {
        int m=i;
        for (int j=i+1;j<n;j++)
            if (a[j]<a[m]) m=j;
        int c=a[i]; a[i]=a[m]; a[m]=c;
    }
}
```

N.B. It takes $n-1$ comparisons to find the minimum of n elements. Thus, the total number of comparisons taken by selection sort is $\sum_{k=2}^{n}(k-1) = n(n-1)/2 = O(n^2)$

Example 3 (Cont'd)

Bubble sort

Version 1 – $O(n^2)$, since $\sum_{k=2}^{n}(k-1) = n(n-1)/2 = O(n^2)$

$$\begin{array}{lcccccccccccc}
i \to 0 & 5 & & & 1 & & & 1 & & 1 & & 1 \\
 & 3 & & i \to 1 & 5 & & & 2 & & 2 & & 2 \\
 & 1 & \Rightarrow & & 3 & & \Rightarrow & 5 & \Rightarrow & 3 & \Rightarrow & 3 \\
 & 4 & & & 2 & & & 3 & i \to n-2 & 5 & & 4 \\
n-1 & 2 & & & 4 & \leftarrow j & & 4 & & 4 & & 5
\end{array}$$

```
void bsort(int n)
{
   for (int i=0;i<n-1;i++)
      for (int j=n-1;j>i;j--)
         if (a[j-1]>a[j]) {
            int z=a[j]; a[j]=a[j-1]; a[j-1]=z;
         }
}
```

Version 2 – $O(n^2)$ in the worst case, e.g. 5 4 3 2 1

$$\begin{array}{lcccccc}
i \to 0 & 1 & & 1 & & 1 & \\
 & 2 & & 2 & & 2 & \\
 & 4 & \Rightarrow & 3 & \Rightarrow & 3 & \text{No swaps, done.} \\
 & 5 & & 4 & & 4 & \\
n-1 & 3 & & 5 & & 5 &
\end{array}$$

```
void bsort(int n)
{
   for (int i=0;i<n-1;i++) {
      bool swapped=false;
      for (int j=n-1;j>i;j--)
         if (a[j-1]>a[j]) {
            int z=a[j];a[j]=a[j-1];a[j-1]=z;
            swapped=true;
         }
      if (!swapped) break;
   }
}
```

Example 3 (Cont'd)

Version 3 – $O(n^2)$ in the worst case, e.g. 5 4 3 2 1

$i \rightarrow 0$	1		1		1
	2		2		2
	5	$\Rightarrow$	3	$\Rightarrow$	3
k	4	$k \rightarrow$	5 $\leftarrow i$		4
$n-1$	3 $\leftarrow j$		4		5

```
void bsort(int n)
{
   for (int i=0;i<n-1;) {
      int k=n-1;                  // Any value ≥ n – 1 will do.
      for (int j=n-1;j>i;j--)
         if (a[j-1]>a[j]) {
            int z=a[j]; a[j]=a[j-1]; a[j-1]=z;
            k=j;
         }
      i=k;
   }
}
```

Insertion sort – $O(n^2)$ in the worst case

Version 1 – Sequential search

$O(n^2)$ comparisons in the worst case, e.g. 5 4 3 2 1
$O(n^2)$ data movements in the worst case, e.g. 5 4 3 2 1

	4		3		1		1		1
$i \rightarrow 1$	3		4		3		3		2
	1	$\Rightarrow i \rightarrow 2$	1	$\Rightarrow$	4	$\Rightarrow$	4	$\Rightarrow$	3
	5		5		5		5 $\leftarrow j$		4
$n-1$	2		2		2	$i \rightarrow n-1$	2		5

1	
3	
4	key
5	
2	

Example 3 (Cont'd)

```
void isort(int n)
{
   for (int i=1;i<=n-1;i++) {
      int j,key=a[i];
      for (j=i-1;j>=0;j--)
         if (a[j]>key) a[j+1]=a[j]; else break;
      a[j+1]=key;
   }
}
```

Version 2 – Binary search

$O\left(\sum_{i=1}^{n-1} \log i\right) = O(\log(n-1)!) = O(n \log n)$ comparisons

$O(n^2)$ data movements in the worst case

$l \rightarrow$	0	1	$l \rightarrow$	0	1		0	1		0	1	
	1	2	$m \rightarrow$	1	2		1	2	$h \rightarrow$	1	2	
	2	4	$h \rightarrow$	2	4	$l,m,h \rightarrow$	2	4	$l \rightarrow$	2	4	
$m \rightarrow$	3	5		3	5		3	5		3	5	
	4	6		4	6		4	6		4	6	
	5	7		5	7		5	7		5	7	key
$h \rightarrow$	6	8		6	8		6	8		6	8	
$i \rightarrow$	7	3	$i \rightarrow$	7	3	$i \rightarrow$	7	3	$i \rightarrow$	7	3	

```
void isort(int n)
{
   for (int i=1;i<=n-1;i++) {
      int l=0,h=i-1;
      while (l<=h) {
         int m=(l+h)/2;
         if (a[i]<a[m]) h=m-1;
         else l=m+1;
      }
      int key=a[i];
      for (int j=i-1;j>=l;j--) a[j+1]=a[j];
      a[l]=key;
}
```

Example 3 (Cont'd)

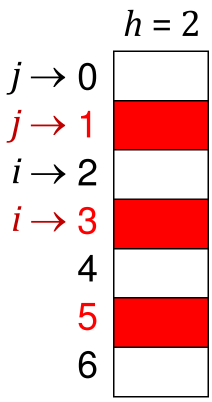
Shell sort

6		1	
5		2	
4	⇒	3	Insertion sort takes 15 comparisons with sequential search.
3		4	
2		5	
1		6	

6		2		1
5		1		2
4	2-step insertion	4	1-step insertion	3
3	⇒	3	⇒	4
2		6		5
1		5		6
	6 comparisons		7 comparisons	

// *h*-step insertion sort; *h* = 1 for insertion sort

```
void h_isort(int n,int h)
{
   for (int i=h;i<=n-1;i++) {
      int j,key=a[i];
      for (j=i-h;j>=0;j-=h)
         if (a[j]>key) a[j+h]=a[j];
         else break;
      a[j+h]=key;
   }
}
```



// $h = 2^k - 1,\ k = \lfloor \log_2 n \rfloor, \ldots, 2, 1;\ \ \text{e.g.}\ n = 31 \Rightarrow h = 15,7,3,1$

```
void shellsort(int n)
{
   for (int k=log((double)n)/log(2.0);k>=1;k--)
      h_isort(n,(1<<k)-1);
}
```

With these increments, it takes a time in $O(n^{1.5})$ in the worst case. It can be made to run in $O(n \log^2 n)$ with a better gap sequence.

Example 4 – Permutation generation

Generate the permutations of $1, 2, \ldots, n$ in lexicographic order (i.e. dictionary order).

Permutation generation algorithm

1. Starting with the smallest permutation $1, 2, \ldots, n$
2. Repeat

   find the next larger permutation

   until the largest permutation $n, \ldots, 2, 1$ is generated.

For example,

1 2 3 → 1 3 2 → 2 1 3 → 2 3 1 → 3 1 2 → 3 2 1

How to find the next larger permutation?

1. Scan the current permutation from right to left

   1.1 It is in increasing order, e.g. 7 6 5 4 3 2 1
   We are done – it is already the largest permutation.

   1.2 If not, locate the 1st number x that is out of increasing order
   e.g. 3 6 2 7 5 4 1

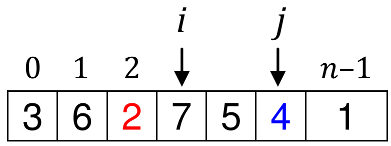
2. Scan from right to left again to find the 1st number $y > x$
   e.g. 3 6 2 7 5 4 1

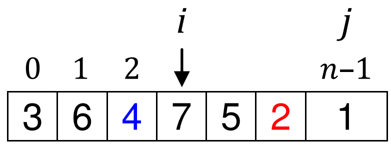
   2.1 Swap x and y, e.g. 3 6 4 7 5 2 1

   2.2 Sort the numbers to the right of y into increasing order
   e.g. 3 6 4 1 2 5 7
   Instead of sorting, this step may be done by swapping:
   3 6 4 7 5 2 1

Example 4 (Cont'd)

```
int a[10];          // at most 10 objects to permute

bool nlp(int n)
{
   int i;
   for (i=n-1;i>0;i--)
      if (a[i-1]<a[i]) break;
   if (i==0) return false;
   int j=n-1;
   while (a[j]<=a[i-1]) j--;
   int z=a[i-1]; a[i-1]=a[j]; a[j]=z;
   j=n-1;
   while (i<j) {
      z=a[i]; a[i]=a[j]; a[j]=z;
      i++; j--;
   }
   return true;
}
```






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

void perm(int n)
{
   for (int i=0;i<n;i++) a[i]=i+1;     // 1,2,...,n
   do out(n); while (nlp(n));
}

int main(void)
{
   perm(3); perm(5);
}
```

Example 5 – Combination generation

Generate all the k-combinations out of $1, 2, \ldots, n$ in lexicographic order, where $k \le n$.

Combination generation algorithm

1 Starting with the smallest combination $1, 2, \ldots, k$

2 Repeat

find the next larger combination

until the largest combination $n-k+1, \ldots, n-1, n$ is generated.

For example, let $n = 5$ and $k = 3$

1 2 3
→ 1 2 4 → 1 2 5 → 1 3 4 → 1 3 5 → 1 4 5
→ 2 3 4 → 2 3 5 → 2 4 5
→ 3 4 5

How to find the next larger combination?

Scan the current combination from right to left

1 Every element attains its maximum value
e.g. 4 5 6 7 8 9 (6-combination out of 9)
We are done – it is already the largest combination.

2 If not, locate the rightmost element that has not yet attained it's maximum value, e.g. 1 2 6 7 8 9

2.1 Increment it by one, e.g. 1 3 6 7 8 9

2.2 Reset all positions to its right to the lowest values possible, e.g. 1 3 4 5 6 7

Example 5 (Cont'd)

```
int a[10];        // at most 10-combinations

bool nlc(int n,int k)
{
   int i;
   for (i=k-1;i>=0;i--)
      if (a[i]!=n-((k-1)-i))
         break;
   if (i<0) return false;
   a[i]++;
   while (i<k-1) {
      a[i+1]=a[i]+1; i++;     // × a[++i]=a[i]+1;
   }
   return true;
}

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

void comb(int n,int k)
{
   for (int i=0;i<k;i++) a[i]=i+1;     // 1, 2, …, k
   do out(k); while (nlc(n,k));
}

int main(void)
{
   comb(6,3); comb(11,10);
}
```