Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

MCS 360 Lecture 21
Introduction to Data Structures
Jan Verschelde, 2 March 2020

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

the factorial of n

Given a natural number n, its factorial $n!$ is

$$n! = n \times (n-1) \times \cdots 2 \times 1.$$

Interpretation: #choices of n items without repetition.
For example: how many 3-letter words with a, b, c?
abc, acb, bac, bca, cab, cba $\#$: $3! = 6$.

What is 0!? How many ways to choose nothing? 0! = 1.

A recursive formula for $n!$ is

$$n! = \begin{cases} 1 & \text{if} \quad n = 0, \\ n \times (n-1)! & \text{if} \quad n > 0. \end{cases}$$

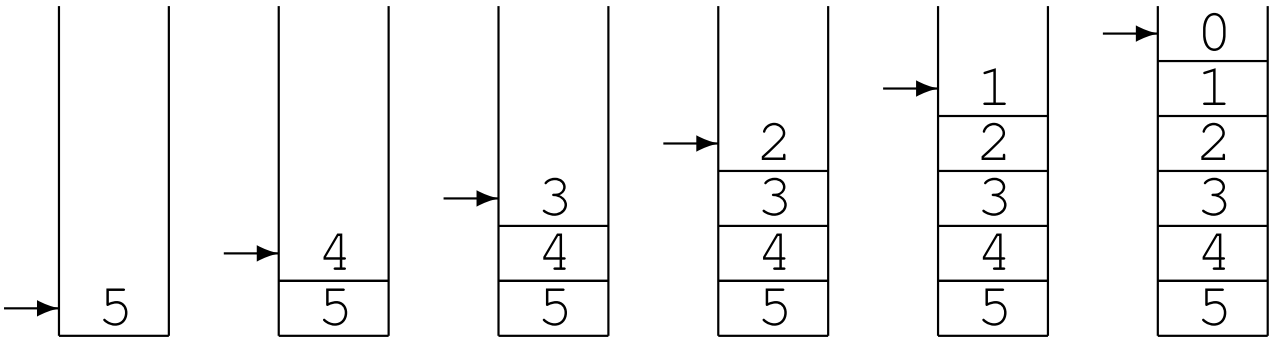
a recursive function

```
int factorial(int n);
// returns the factorial of n

int factorial(int n)
{
   if(n==0)
      return 1;
   else
      return n*factorial(n-1);
}
```

stack of function calls

Computing 5! recursively happens via a stack



- not in the base case: push argument on the stack
- after base case, pop from stack and evaluate

unwinding the recursion

factorial(5)
→ factorial(4)
→ factorial(3)
→ factorial(2)
→ factorial(1)
→ factorial(0)
return `1`
return `1*1`
return `2*1*1`
return `3*2*1*1`
return `4*3*2*1*1`
return `5*4*3*2*1*1`

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

tracing recursion

```
tracing 5! ...
      n = 5
     n = 4
    n = 3
   n = 2
  n = 1
n = 0
 returning 1
  returning 2
   returning 6
    returning 24
     returning 120
```

the function `trace_factorial`

```
int trace_factorial(int n)
{
   for(int i=0; i<n; i++) cout << " ";
   cout << "n = " << n << endl;

   if(n==0)
   {
      return 1;
      cout << "returning 1" << endl;
   }
   else
   {
      int r = n*trace_factorial(n-1);
      for(int i=0; i<n; i++) cout << " ";
      cout << "returning " << r << endl;
      return r;
   }
}
```

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

an accumulating parameter

Instead of a tail recursion for $n!$,
we can accumulate the result in a parameter.

```
int accumulate_factorial(int n, int f)
{
   if(n <= 1)
      return f;
   else
      return accumulate_factorial(n-1,n*f);
}
```

tracing the execution

```
computing 5*1
computing 4*5
computing 3*20
computing 2*60
returning 120
```

```
int trace_accumulate_factorial(int n, int f)
{
  if(n <= 1)
  {
    cout << "returning " << f << endl;
    return f;
  }
  else
  {
    cout << "computing " << n << "*" << f << endl;
    return trace_accumulate_factorial(n-1,n*f);
  }
}
```

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

generating n random numbers

A recursive view of a nonempty list `L`:
`L` has a node as head and a list as tail.

A recursive algorithm to generate n numbers:

- If n equals zero (or less) then

  return an empty list;

- else (n is larger than zero)
  - generate a list `L` of n-1 numbers;
  - push a random number to `L`.

the function generate

```
list<int> generate ( int n )
{
   if(n <= 0)
   {
      list<int> L;
      return L;
   }
   else
   {
      int r = rand() % 1000;
      list<int> L = generate(n-1);
      L.push_front(r);
      return L;
   }
}
```

tracing the execution

Making `generate` verbose with print statements:

```
generating 3 random numbers ...
generate with n = 3 ...
generate with n = 2 ...
generate with n = 1 ...
generate with n = 0 ...
returning empty list
pushing 73 to front ...
pushing 249 to front ...
pushing 807 to front ...
```

What is the content of the list on return?

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

writing a list recursively

A recursive algorithm to write a list `L`:

- if list `L` is empty then

  we do nothing;

- else (`L` is not empty)
  - pop first item `i` from `L`;
  - write `i`;
  - write `L`; // we have popped `i`

the function `write`

```
void write ( list<int> L )
{
   if(!L.empty())
   {
      list<int> K = L;
      cout << " " << K.front();
      K.pop_front();
      write(K);
   }
}
```

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

searching a list recursively

Does a number `e` belong to a list `L`?

- if the list `L` is empty then

  return false;

- else if front of `L` equals `e` then

  return true;

- else
  - pop front element from list `L`;
  - return belongs `e` to `L`? // `L` is smaller

the function `belongs`

```
bool belongs ( list<int> L, int e )
{
   if(L.empty())
      return false;
   else if(L.front() == e)
      return true;
   else
   {
      list<int> K = L;
      K.pop_front();
      return belongs(K,e);
   }
}
```

Recursive Definitions

1. Recursive Mathematical Formulas
   - the factorial of a natural number
   - tracing a recursive execution
   - an accumulating parameter
2. Recursion on STL Lists
   - generating n random numbers
   - writing a list recursively
   - searching a list
3. Recursive Greatest Common Divisor
   - computing the greatest common divisor recursively

the greatest common divisor

Observe: gcd(20,15) = gcd(15,5), 5 = 20 % 15.

A recursive algorithm to compute gcd(a,b):

- base case: if(a % b == 0)
  then b divides a, so b == gcd(a,b)
- else, let r = a % b, return gcd(b,r).

Why is this an algorithm?

- termination: r < b; if b > a, then r = a
- correctness: gcd(a,b) == gcd(b,a % b)

the function `gcd`

```
int gcd(int a, int b)
{
   int r = a % b;
   if(r == 0)
      return b;
   else
      return gcd(b,r);
}
```

tracing the execution

```
give x : 98212
give y : 44632
gcd(98212,44632) = 4
tracing gcd(98212,44632) :
a = 98212, b = 44632, r = 8948
a = 44632, b = 8948, r = 8840
a = 8948, b = 8840, r = 108
a = 8840, b = 108, r = 92
a = 108, b = 92, r = 16
a = 92, b = 16, r = 12
a = 16, b = 12, r = 4
a = 12, b = 4, r = 0
```

Summary + Exercises

Started Chapter 7 on recursive algorithms.

Exercise 1:

1. Write the definition of a function to copy the elements on a stack ***recursively*** to a list.

   Use the prototype below.

```
list<int> copy ( stack<int> stk );
// Returns a list with the same elements on the
// stack stk.
// The elements in the returned list occur in
// the same order as on the stack, the top of
// the stack occurs first; the bottom of the
// stack is the last element in the list.
```

more exercises

Additional exercises:

2. Draw the evolution of the stack of function calls for 5! computed recursively with an accumulating parameter.
3. Write a recursive function to sum a STL list of integer numbers. Give code to show that your function works.
4. Test what happens when the arguments for the gcd function would be negative numbers.