CS 126 Lecture P6: Recursion

Why Learn Recursion?

• Master a powerful programming tool
• Gain insight of how programs (function calls) work
Outline

- **What is recursion?**
- How does it work?
- Examples
Number conversion

- To convert an integer \( N \) to binary:
  - stop if \( N \) is 0
  - write “1” if \( N \) odd, “0” if \( N \) even
  - move left one position
  - convert \( N/2 \)

**Ex:**
- 42 0
- 21 10
- 10 010
- 5 1010
- 2 01010
- 1 101010

**check:**
- 5 4 3 2 1 0
- 1 0 1 0 1 0
- 2 1
- 32 + 8 + 2 = 42

- Easiest way to convert to binary by hand
- Corresponds directly to a recursive program

Recursive number conversion

- Computer prints from left to right
- need to convert \( N/2 \), then print right bit

```c
void convert(int N)
{
    if (N/2 > 0) convert(N/2);
    printf("%c", '0' + N % 2);
}
```

**Proof of correctness:** \( N = 2\times(N/2) + (N \% 2) \)

- `convert(42)`
- `convert(21)`
- `convert(10)`
- `convert(5)`
- `convert(2)`
- `convert(1)`
- `printf("1")`
- `printf("0")`
- `printf("1")`
- `printf("0")`
- `printf("1")`
- `printf("0")`

- Works to convert to any base
  (change “2” to “b” everywhere in code)

Indentation level denotes statements belonging to same “invocation”
Outline

• What is recursion?
• How does it work?
• Examples
Function “Environment”

• When a function executes, it lives in an “environment”
• What’s an “environment”?
  - Value of local variables (scratch space)
  - Which statement the computer is executing currently

Implementing Recursion

*Any* function call requires the system to
  • set the values of the parameters
  • save the “environment”
  • jump to the first instruction in the function
  • execute the function
  • restore the “environment”
  • continue at the instruction after the call
    “return address” (part of environment)

- Use pushdown stack for save/restore
  call: push environment
  return: restore environment from stack
Demo Use of Stacks to Implement Function Calls
Removing Recursion

We can remove recursion from any function by using an explicit stack.

Helps us understand nature of the computation (no other reason to do so)

Example: number conversion

```c
void convert(int N)
{
    STACKinit();
    while (N > 0) { STACKpush(N); N = N/2; }
    while (!STACKempty())
    {
        printf("%c", '0' + STACKpop() % 2);
    }
}```
Tail Recursion

- If single recursive call is the last action, don’t need a stack
- Why?
  - nothing to do after recursion => no need to remember stuff => no need for stack

Possible Pitfall with Recursion

Simple recursive programs can consume excessive resources

Ex: Compute binomial coefficients

```c
int f(int N, int k)
{
    if ((k < 0) || (k > N)) return 0;
    if (N == 0) return 1;
    return f(N-1, k) + f(N-1, k-1);
}
```

Q: Why?
A: Recomputes intermediate results
Outline

- What is recursion?
- How does it work?
- Examples
Divide-and-Conquer

Many computations are naturally expressed as recursive programs.

ITERATION
another way to write "for" loop

"DIVIDE and CONQUER"
solve a problem by dividing into smaller ones

Ex: root finding via "bisection"

Finding Root via Bisection

```c
float bisectr(float l, float r)
{
    float m;
    m = (l + r) / 2;
    if ((r - l) < epsilon) return m;
    if (f(m) > 0.0)
        return bisectr(m, r);
    else
        return bisectr(l, m);
}
```
Bisection for Integer Functions

```c
int bisectr(int l, int r)
{
    int m;
    m = (l + r) / 2;
    if (f(m) == 0) return m;
    if (r <= l) return -1;
    if (f(m) > 0)
        return bisectr(m+1, r);
    else
        return bisectr(l, m-1);
}
```

Binary Search

- Observations:
  - An array is a function mapping integer indices to contents
  - A sorted array is a monotonically increasing function
Traveling Salesman Problem

Given a set of points, find the shortest tour connecting all the points

- Recursive solution for trying all possibilities

```c
visit(int N)
{
    int i;
    if (N == 1) { checklength(); return; }
    for (i = 1; i <= N; i++)
    {
        swap(i, N);
        visit(N-1);
        swap(i, N);
    }
}
```

Visit ith city as the last (Nth) step

Decide the positions of the other undecided cities

Number of nodes whose positions have not been decided
Traveling Salesman Problem

- visit(3) [1 2 3]{}
Intuition of Algorithm

• $AB$ is a smaller dragon curve by itself
• $CB = AB$
• Therefore $BC$ is the reverse of $AB$
• Therefore every turn along $BC$ is the opposite of the corresponding turn on $AB$
Recursive Program for Dragon Curve

dragon(int n)
{
    if (n == 0) { F(); return; }
    dragon(n-1);
    L();
    nogard(n-1);
}

Backwards Dragon Curve

dragon(int n)                    nogard(int n)
{
    if (n == 0) { F(); return; }  if (n == 0) { F(); return; }
    dragon(n-1);
    L();
    nogard(n-1);
    R();
    Reverse
    nogard(n-1);
}
dragon Demo

Alternate "dragon"

- Replace call to 'nogard' by nonrecursive version

```c
int dragon(int n)
{
    int k;
    if (n == 0) { F(); return; }
    dragon(n-1);
    L();
    for (k = n-2; k >= 0; k--)
    {
        dragon(k);
        R();
    }
    F();
}
```

Points out self-similarities in curve
Postscript dragon curve

- Easy to implement because of built-in
- turtle graphics
- stack
- Passing args to recursive functions is tricky
  - replicates top before popping for comparison
  - pushes two copies of (n-1) for the two recursive calls

```
/L { 90 rotate } def
/R { 90 neg rotate } def
/F { 2 0 rlineto } def
dragon
{ dup 0 eq
  { F pop }
  { 1 sub dup dragon L nogard }
} ifelse
} def
/nogard
{ dup 0 eq
  { F pop }
  { 1 sub dup dragon R nogard }
} ifelse
} def
200 400 moveto
15 dragon
stroke
showpage

CAUTION:
$2^N$ line segments in curve of order $N$.
```

Nonrecursive dragon curve

To write down the whole dragon curve sequence
- first, put "F" in every other space
- put "L", "R" (alternating) in every other remaining space
- continue until done

```
F F F F F F F F
F L F F F F F F F F
F L F F F F F F F F F F
F L F F F F F F F F F F F F
```

- Like Towers of Hanoi (see Sedgewick, section 5.1)
  - requires too much storage (how much?)
  - "ruler function" connects to binary numbers
  - details? [challenge for the bored]

```
step 1: L if the bit to the left of the rightmost 1 in the binary rep. of $n$ is 1
```
What We Have Learned

• How recursion works
  - A recursive call is no different from a “regular” call
  - It involves saving the old environment for later return

• Learn to trace the execution of given recursive programs (using pictures)

• Learn to write simple recursion
  - What’s the base case?
  - What’s the induction case?