CS 52 : C++ Programming

Howard A. Stahl
Agenda

• Recursion

• Arrays
  – Array Parameters
  – Typical Array Operations
What Is Recursion?

• Sometimes, The Best Way To Solve A Problem Is To Solve A Smaller Problem Of The Exact Same Type
• Recursion Is The Technique Of Solving A Problem In Terms Of Itself
What Is Recursion?

• A Recursive Definition Uses The Term We Are Defining To Define The Term
  – has the form: \( q = \ldots q \ldots q \ldots q \ldots \)

• Examples:
  – \( x = 12x + x^x \)
  – A descendant is an offspring of the person or a descendant of an offspring of a person
  – \( n! \)
Cases In Recursion

• Base Cases - Not Involved In Recursion
  – descendant: offspring of the person

• Recursive Cases -
  – descendant: descendant of an offspring
Recursive Functions

• Factorial
  – \( 0! = 1 \)
  – \( n! = n \times (n-1)! \) for \( n > 0 \)

• Compound Interest
  – \( A(0) = P \)
  – \( A(n) = (1+r)A(n-1) \) for \( n > 0 \)
Recursive Programming

• Functions Defined By Calling Themselves

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

```c
int factorial( int n ) {
    fac = 1;
    for (int i = 1; i <= n; i++) fac = fac * i;
    return( fac );
}
```
Recursive Programming

5!
5 * 4!
4 * 3!
3 * 2!
2 * 1!
1 * 0!
1

120
5 * 24
4 * 6
3 * 2
2 * 1
1 * 1
Rules Of Recursive Programming

• Base Cases
  – cases solved without recursion

• Make Progress
  – recursive cases should make progress toward a base case

• General Case
  – assuming recursive calls work, make the whole function work
Example: Fibonacci Sequence

• An Italian (1170-1250AD) Posed The Following Puzzle:
  – A Man has an infant male-female pair of rabbits in a hutch entirely surrounded by a wall. We wish to know how many rabbits can be bred from this pair in one year, if every month they breed one other male-female pair which themselves begin to breed in the second month after their birth. Assume no deaths during the year.
Diagram Of Fibonacci’s Puzzle

Number of Pairs
1
1
2
3
5
Observations

• The number of rabbits at the beginning of any month equals the number of rabbits of the previous month plus the number of new pairs

• The number of new pairs at the beginning of a month equals the number of rabbits two months ago
Observations

• The number of rabbits at the beginning of any month equals the number of rabbits of the previous month plus the number of new pairs.

• The number of new pairs at the beginning of a month equals the number of rabbits two months ago.

• The Sequence: 1,1,2,3,5,8,13,21, ...... , 233
Fibonacci Sequence

• Recursive Definition
  – $f(0) = 1$
  – $f(1) = 1$
  – $f(n) = f(n-1) + f(n-2)$
Fibonacci Function

// precondition: n must be > 0
int Fibonacci( int n ) {
    // BASE CASE: 0
    if (n == 0) return( 1 );
    if (n == 1) return( 1 );
    return( Fibonacci( n-1 ) + Fibonacci( n-2 ) );
}
Recursion Is Neat...

• But It Doesn’t Always Apply
Recursion Is Neat...

- But It Doesn’t Always Apply
- Consider A Dinner Party Algorithm
Recursion Is Neat...

• But It Doesn’t Always Apply
• Consider A Dinner Party Algorithm
  – MakeDinner( # Guests ) =
    MakeDinner( 1 ) +
    MakeDinner( # Guests - 1 )
Recursion Is Neat...

• But It Doesn’t Always Apply
• Consider A Dinner Party Algorithm
  – MakeDinner( # Guests ) =
    MakeDinner( 1 ) +
    MakeDinner( # Guests - 1 )
• But Would AnyOne Do It This Way???
• Recursion Can Typically Be Written Iteratively, So Don’t OverDo It!
Recursion vs. Iteration

• Iteration Can Be Used In Place Of Recursion
  – recursion uses a branching construct
  – iteration uses a looping construct

• Recursive Solutions Are Often Less Efficient, In Terms Of Both Time And Space, Than An Iterative Solution

• Recursive Solutions Can Simplify The Solution Of A Problem
How Is Recursion Implemented?

• Function Calls Result In Activation Records Being Recorded On A Run-Time Stack
  – the computer has to stop executing function a and start executing function b
  – since it must come back to function a later on, it needs to store everything about function a, including where to pick up when function b returns
  – bind values to function b parameters
  – transfer control to function b
How Is Recursion Implemented?

• After Function B Finishes, The Activation Record Is Popped Off The Stack
• All The Old Values Are Restored In The Context Of Function A
• Assign The Return Value Of The Function
• Transfer Control Back To Where It Left Off
How Is Recursion Implemented?

• Apart From The Fact That The Caller And Callee Have The Same Name, There Is No Difference Between The Way Recursive And Non-Recursive Calls Are Handled
Time For Our First Demo!

• Fibonacci.cpp

(See Handout For Example 1)
Summarizing Our First Demo!

- Recursion Can Create Very Natural Algorithms
Arrays

• We Typically Encounter Groups Of Like-Minded Objects
  – eggs in an egg carton
  – apartments in an apartment building
Arrays

• We Typically Encounter Groups Of Like-Minded Objects
  – eggs in an egg carton
  – apartments in an apartment building
• Each Object In The Set Is The Same
• The Overall Set Has A Size
Arrays

- We Typically Encounter Groups Of Like-Minded Objects
  - eggs in an egg carton
  - apartments in an apartment building
- Each Object In The Set Is The Same
- The Overall Set Has A Size
- C++ Has A Similar Construct
  - arrays
Arrays

• An Array Is A Collection Of Values Of All Identical Type
  – classes also contain collections of values, but these values are of different types

• The Collection Has A Variable Name

• Each Item In The Collection Has A Subscript That Defines Its Position
Array Declaration

• Syntax:
  
  ```
  type arrayname[size];
  ```

• `type` referred to as the base type for all array elements

• `arrayname` is the variable name for the entire collection

• `size` is the number of elements allowed in the collection
  – indexes from 0 to `size-1`
Arrays

• Example:
  - int grades[ 5 ];
Arrays

- Example:
  ```c
  int grades[5];
  ```

Each Indexed Element Is An int
Indexes Start At Zero
Arrays

• Arrays Are An Ordered List

• Arrays Are Stored Contiguously In One Block

• Each Index Is An lvalue In Its Own Right

• [ ] Is Used To Declare And Access Arrays
Arrays

• Example:

```cpp
int grades[3];
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```
Arrays

• Example:

```c
int grades[3];
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```
Arrays

• Example:

```java
int grades[3];
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```

Don’t Confuse Declaration Syntax With Element Access
Array Initialization

• Like Other Variables, Arrays Can Be Initialized When They Are Declared
• Generally, It’s A Good Idea To Define Constants For Array Size

```c
const int SIZE=3;
int grades[SIZE];
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```
Array Initialization

• Like Other Variables, Arrays Can Be Initialized When They Are Declared

• Generally, It’s A Good Idea To Define Constants For Array Size

```c
const int SIZE=3;
int grades[SIZE];
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```
Array Initialization

• Like Other Variables, Arrays Can Be Initialized When They Are Declared

• Generally, It’s A Good Idea To Define Constants For Array Size

```c
const int SIZE=3;    const int SIZE=3;
int grades[SIZE];    int grades[SIZE]={1,10,100};
grades[0] = 1;
grades[1] = 10;
grades[2] = 100;
```
Array Iteration

- for Loops Are Often Used With Arrays
  - array index need not be a fixed constant

```cpp
const int SIZE=3;
int a[SIZE]={1,10,100};

for (int i=0; i<SIZE; i++) {
    cout << "a[" << i << "]=" << a[i] << endl;
}
```
Important Considerations

• Don’t Exceed Array Bounds
  – OutOfBounds Errors Cause Problems
• Typically, Bounds Errors Come On The Last Iteration Going Over The Edge
• You Are Forewarned!
Time For Our First Demo!

• ArrayCode.cpp

(See Handout For Example 1)
Summarizing Our First Demo!

- Arrays Let You Work With Groups Of Data
- Carefully Track Array Size!
Arrays As Function Parameters

• Like Any Other \textit{lvalue}, Array Elements Can Be Passed To Functions

```c
void print_value( int i );

const int SIZE=3;
int a[SIZE]={1,10,100};

for (int i=0; i<SIZE; i++) {
    print_value( a[ i ] );
}
```
Arrays As Function Parameters

• The Whole Array Can Also Be A Parameter To A Function

• Arrays Are Passed To Functions As Array Parameters
  – neither pass-by-value or pass-by-reference
  – closely mimics pass-by-reference

• If A Function Changes Element Value, These Changes Will Be Seen By The Caller
Arrays As Function Parameters

- Formal Parameter Syntax: `type name[]`
- Actual Parameter Syntax: `name`
Arrays As Function Parameters

- Formal Parameter Syntax: `type name[]`
- Actual Parameter Syntax: `name`

```c
void fill_up( int items[], int length );

const int SIZE=3;
int a[SIZE]={1,10,100};

fill_up( a, SIZE );
```
Arrays As Function Parameters

- **Formal Parameter Syntax:** `type name[]`
- **Actual Parameter Syntax:** `name`

```c
void fill_up( int items[], int length );

const int SIZE=3;
int a[SIZE]={1,10,100};

fill_up( a, SIZE );
```
Observations

• Since The Array Parameter Definition Lacks Array Size Value, It Is Always A Good Idea To Pass The Size Of The Array As An Extra Argument
Observations

- Since The Array Parameter Definition Lacks Array Size Value, It Is Always A Good Idea To Pass The Size Of The Array As An Extra Argument

```c
void fill_up( int items[], int length );

int a[5], b[10];

fill_up( a, 5 );
fill_up( b, 10 );
```
Observations

• Since The Array Parameter Definition Lacks Array Size Value, It Is Always A Good Idea To Pass The Size Of The Array As An Extra Argument

```c
void fill_up( int items[], int length );

int a[5], b[10];

fill_up( a, 5 );
fill_up( b, 10 );
```

Function Can Be Called With Arrays Of Various Sizes
Observations

• When Arrays Are Passed To Functions, Elements Changed By The Function Are Visible To The Caller

• Array Parameters Are Kinda Pass-by-Reference
  – No copies of the individual elements are made
  – Changes to any elements will be seen by the caller
const Array Arguments

- If you know the function will not change the array values, use `const` modifier
const Array Arguments

• If You Know The Function Will Not Change The Array Values, Use const Modifier

void print(const int items[], int length);
Further Complexities

• Arrays Can Be Have Classes As Elements
  – class MUST have a default (parameterless) constructor
  – gets invoked for each element

• Classes Can Have Arrays As Members

• Arrays May Be Partially Filled
  – the first $x$ elements contain values
  – the ending $y$ elements are empty
Typical Array Operations

• Searching
• Sorting
Time For Our Next Demo!

• ArrayRecursiveCode.cpp

(See Handout For Example 3)
Summarizing Our Next Demo!

• Array Operations Can Get Quite Expensive As The Array Grows In Size

• Recursion Can Be Used In Place Of for-loop Processing Of Array Elements
Summary

• Recursion
• Arrays
  – Array Parameters
  – Typical Array Operations