CSE 12, Week Six, Lecture Two

Recursion:
What:
- A method that calls itself

When:
- Need to solve a loop-like problem
- When we have a problem that can be solved through a loop, where the problem simplifies with each iteration: choice is loop or recursion.

How:
- 1. Write code to solve one part of your problem.
- 2. Call a method to perform the rest of that task.
- 3. The method you call is the same one you are writing
- 4. Need a recursion terminating condition.

Why:
- Once you understand recursion, the solution is implemented in less code that a looped based solution.
- Less code = less time = less bugs

Mechanism:
- Before a function call begins executing, the “return address” is placed on the stack.
- “Return address”: an address into the “text” section where program executing is to resume once that function ends.
- Often the return address is commingled with user data (parameters, local variables).
| Stack frame for last recursive decout call | remainder | 1 |
|                                           | quotient  | 0 |
|                                           | number    | 1 |
|                                           | return address | $ 500 |
| Stack frame for next recursive decout call | remainder | 2 |
|                                           | quotient  | 1 |
|                                           | number    | 12 |
|                                           | return address | $ 500 |
| Stack frame for first recursive decout call | remainder | 3 |
|                                           | quotient  | 123 |
|                                           | number    | 123 |
|                                           | return address | $ 500 |
| Stack frame for initial decout call       | remainder | 4 |
|                                           | quotient  | 123 |
|                                           | number    | 1234 |
|                                           | return address | $ 1000 |

<table>
<thead>
<tr>
<th>Label/description</th>
<th>Value on RTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: <em><strong>1_2_3_4</strong></em>____ (as each stack frame is popped).</td>
<td></td>
</tr>
</tbody>
</table>
Implications:
- Parameters are distinct from original variable that is passed to function call: call by _____
- Local variables are created ____________________
- With recursion local variables exist ______ times, one copy for each ___________________.
- Debugger/Java exceptions use the RTS ____________________
- Infinite recursion will ____________________.
- Overwriting a RTS array could mean overwriting return addresses, and then you can’t ____________________.
- Parameters act just like ________________.
- More resources within your allotted resources are consumed with recursion than with a loop, those resources (__________) are allocated to you when ________________.
- RTS allocation is the ________ efficient allocation possible:
  o Allocation and decallocation occur with assignment of ________________:
    ▪ PC is Program Counter: address in the text section for the instruction about to execute.
    ▪ SP is the Stack Pointer: address in the RTS that points to the top of the RTS.
      • Call resets ____, allocating __________
      • Call resets ____, which _____________
      • Return: restores the ___
      • Return: stores _______________
Constants:
- The value associated with symbol can ____ change.
- Syntax: “const” keyword
- Parameters:
  o Parameter is read-only
    ▪ Prefix “const” to the parameter declaration.
    ▪ Enforced in method and in any methods called using parameter.
    ▪ Originating value may or may not be const (mostly not).
    ▪ Look but don’t touch.
    ▪ Provided documentation to the user giving assurance that original item passed won’t change.
    ▪ When to use:
      • ____________________!
- Member methods:
  o Object whose member method is being called is read only.
    ▪ Add “const” after parameter list before open curly brace to the method body.
    ▪ When to use:
      • ____________________!
- Return result:
  o Result is read-only
    ▪ Prefix “const” to the return type of the method.
    ▪ When to use:
      • When object is still needed in its original form elsewhere.
Access rights:
- What: How to restrict access to data and method fields of your objects.
- Allows: More design choices for software engineer.
- Implements: data hiding, encapsulation.
- Applies to:
  - Sections of classes (C++)
  - Individual class data and method fields (Java)

Inaccessible section:    // inaccessible by __________
- no keyword
- private section of an object ____ accessible to any members or friends of that object.
- private section of the parent object from the perspective of the child object.
  - Ex: _______________________

Private:
- keyword “private”
- accessible to members and friends.

Protected:
- keyword “protected”
- _______ section of the parent accessible to _______.
  - Ex: ____________

Public:
- keyword “public”
- accessible by all.
Derivations:
What: creating a new type from an existing type.
- All fields of the parent begin a child object.
  o Child object can add more fields.
C++: Three kinds of derivations:
- private
- protected
- public
Java:
- like a public derivation

Syntax:
Java:
  class Child extends Parent {…}

C++:
  class Child : private Parent {…};
  class Child : protected Parent {…};
  class Child : public Parent {…};

C++ legal but rarely used:
  class Child : Parent {…};
  - ________derivation if Parent is a class
  class Child : Parent {…};
  - ________derivation if Parent is a struct
What are the access rights on the fields of the parent when existing in a child object?

Themes:
- Only restrict access or allow the same level of access.
- Never give more access in the child than the parent had.
- The parent has access to its own fields, even in a child object.

Desire: All fields of the parent have the same access rights in the child as the type of derivation:

Columns: Derivation type
Rows: Section of the child originating from the respective section of the parent.

<table>
<thead>
<tr>
<th>private</th>
<th>protected</th>
<th>public</th>
</tr>
</thead>
<tbody>
<tr>
<td>inaccessible</td>
<td></td>
<td></td>
</tr>
<tr>
<td>private</td>
<td></td>
<td></td>
</tr>
<tr>
<td>protected</td>
<td></td>
<td></td>
</tr>
<tr>
<td>public</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Constructor rules:
- Don’t have a return type.
- More than one can exist per class.
  - Distinguished by parameter list (overloading)
- Called when an instance comes into existence:
  - RTS objects:
    - ________________:
      - Local objects (default or overloaded constructor)
    - Parameter objects (_____ constructor)
      - Reference parameters are not new objects
        - _____ constructors are called for reference parameters.
    - Return result objects (_____ constructor)
      - Reference results are not new objects
        - _____ constructors are called for reference results.
  - Heap objects:
    - ______________________
  - Data objects:
    - ______________________!
Default constructor:

- Used to create an object without passing parameters to constructor.
- Prototype syntax: Classname (void);
- Sometimes the default constructor is optional:
  o If all instances are created by passing parameters to constructors.
    ▪ Default constructor is optional since it was never called.
  o If all instances are created without passing parameters to constructors.
    ▪ No constructors are ever called.
    ▪ Therefore, no constructors need to exist!
    ▪ Instantiated object was created not constructed.
- Sometimes the default constructor is mandatory:
  o If instances are created both by passing parameters and without passing parameters to constructors.
    ▪ One you define the non-default constructor, you must define default one, too (as long as you are creating default objects).
Copy constructor:
- Used to create an object from another object.
- Prototype syntax: Classname (Classname &);
- Implicitly called:
  o Creating an object as a parameter to a method.
  o Creating an object as a return result from a method.
- Always optional:
  o Default behavior if missing:
    ▪ Member-wise copy is made.
    ▪ New object created is distinct from the original if the object is flat (no pointer data fields).
    ▪ Instantiated object was created not constructed.
Destructor rules:
- Don’t have a return type.
- Only one can exist per class.
  - No parameters can be passed to it.
- Used to destroy or clean up when an instance goes out of scope.
- Prototype syntax: ~Classname (void);
- Always optional:
  - Okay to not have one when the object is flat.
  - If pointer fields exist, then you’ll need to deallocate memory with “delete” inside the body of destructor.
    - Otherwise: _______________
- Implicitly Called:
  - When a constructed object goes out of scope.
    - RTS objects:
      - ________________
    - Heap objects:
      - ________________
    - Data objects:
      - ________________
- Can be explicitly called:
  - Code of destructor executed like any other method.
    - Can be used to re-initialize an object.
    - Rarely done.

Destroying an array of heap objects:
- Array brackets are needed to let the compiler know to call the destructor on all instances in the array.
- Syntax: delete [] array_name;