CS 31: Introduction To Computer Science I
Howard A. Stahl

Agenda
• void Functions
• return Statement
• PostFix and Prefix ++, -- Operators
• Default Value Arguments
• Parameter Passing Mechanisms
• Problem Solving and Testing Strategy

void Functions
• Functions Need Not Always Return A Result
• A Return Type Of void Indicates A Function That Returns No Result
  - return statement contains no expression
  - return statement assumed at end of function
• In Other Languages, void Functions Are Called Subroutines
void Functions

• Examples:
  void horizontal_line( ) {
    cout << "\n-------------\n";
    return;
  }
  void sayGoodnightGracie( ) {
    cout << "Goodnight, Gracie";
  }

return Statement

• A Function May Contain Multiple return Statements
  int max( int a, int b) {
    if (a < b)
      return b;
    else
      return a;
  }
• Generally, More Readable With Just One
Prefix and Postfix Operators

- `++` is a shorthand for `+ 1`:
  
  ```
  i++;      i = i + 1;
  ```

- `--` is a shorthand for `- 1`:
  
  ```
  i--;      i = i - 1;
  ```

- The operator can come before or after the variable:
  
  ```
  --i++;      ++i;
  ```

Prefix and Postfix Operators

- Prefix operator occurs before expression evaluation
- Postfix operator occurs after statement evaluation

```java
int i = 12, j = 10, k = 0;
k = i++ * --j;
k = --i + ++j;
```

Default Valued Arguments

- Functions can have “optional” arguments
- They are defined, but do not need to be passed by the caller
Default Value Arguments

- Functions Can Have “Optional” Arguments
- They Are Defined, But Do No Need To Be Passed By The Caller
- If Not Passed, A Default Value Will Be Supplied Automagically
- Default Valued Arguments Must Be Grouped Together At The End Of The Parameter List
Parameter Passing

• So Far, Our Functions Cannot Alter Their Parameters
  – referred to as “pass-by-value”
  – these functions can only provide a single output value
• However, There Is Another Kind Of Parameter Passing Scheme
  – referred to as “pass-by-reference”

Parameter Passing

• Reference Parameters Are Not Copies Of The Actual Parameter, But Are The Parameters Themselves
• Actual Parameters Must Be A Variable
  – referred to as an "lvalue", as opposed to an "rvalue"
• Specified When The Prototype Use The Syntax: `type&`
  – recall from C that & means “address of”

Call-By-Reference Example

```c
// A simple example of call-by-reference parameters
3
#include <stdio.h>
1

void procedure(int*, int*);
2

int main()
3
{
4
    int x = 10, y = 20;
5
    procedure(&x, &y);
6
    printf("x = %d, y = %d\n", x, y);
7
    return 0;
8
}
9

void procedure(int* a, int* b)
10
{
11
    *a = 30;
12
    *b = 40;
13
}
14
```
Call-By-Reference Example

```cpp
18 void getNumbers(int& input1, int& input2)
19 { 
20   cout << "Enter two integers: ";
21   cin >> input1;
22   >> input2;
23 }
24 void swapNumbers(int& variable1, int& variable2)
25 { 
26   int temp;
27   temp = variable1;
28   variable1 = variable2;
29   variable2 = temp;
30 
31 void shuffleData(int output1, int output2)
32 { 
33   cout << "In reverse order the numbers are: ";
34   output1 = " " << output2 << endl;
35 }
36 
```}

Call-By-Reference Example

Parameter Passing

- Reference Parameter Example:
  ```cpp
  void swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
  }
  ```

- Legal Invocation???
  ```cpp
  int i=0, j=20;
  swap( i, j );
  ```
Parameter Passing

• Reference Parameter Example:
  ```c
  void swap(int& x, int& y) {
    int temp = x;
    x = y;
    y = temp;
  }
  ```

• Legal Invocation???
  ```c
  int i=0, j=20;
  swap( i, j++ );
  ```

Function Call And Return

```c
void swap( int &x, int &y);
main( )
{
  int i = 0, j = 20;
  swap( i, j );
  return 0;
}
```
Function Call And Return

```c
void swap(int &x, int &y);

main()
{
    int i = 0, j = 20;
    swap(i, j);
    return 0;
}
```

Memory Model

<table>
<thead>
<tr>
<th>1000</th>
<th>1004</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>j</td>
</tr>
</tbody>
</table>

```c
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```
Function Call And Return

void swap(int &x, int &y);
main()
int i = 0, j = 20;
swap(i, j);
return 0;

Memory Model
1000 0 i
1004 20 j

void swap(int &x, int &y)
int temp = x;
x = y;
y = temp;

void swap(int &x, int &y);
main()
int i = 0, j = 20;
swap(i, j);
return 0;

Memory Model
1000 0 i
1004 20 j

void swap(int &x, int &y)
int temp = x;
x = y;
y = temp;

void swap(int &x, int &y);
main()
int i = 0, j = 20;
swap(i, j);
return 0;

Memory Model
1000 0 i
1004 20 j

void swap(int &x, int &y)
int temp = x;
x = y;
y = temp;
Function Call And Return

```c
void swap(int &x, int &y);

main()
{
    int i = 0, j = 20;
    swap(i, j);
    return 0;
}
```

```c
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

Memory Model

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>20</td>
</tr>
<tr>
<td>1004</td>
<td>0</td>
</tr>
</tbody>
</table>

Summarizing Parameter Passing

- The Caller Passes The Address Of Actual Reference Parameters To Invoked Functions
Time For Our First Demo!

- Reference.cpp

(See Handout For Example 1)

Summarizing Our First Demo!

- Pass-By-Value Results In Copies Being Made Of Every Argument
  - this might have a performance impact on your code
- However, Pass-By-Reference Makes Things More Complex
  - your function may have unintended side effects, since it can change values inside the caller’s world

Mixing Parameter Types

- A Function May Use Both Kinds Of Parameter Passing Schemes In One Prototype
  
  ```
  void process(int input, int& output);
  ```

  this parameter passed by value
  this parameter passed by reference
Problem Solving Strategy

- One big problem is harder to solve than many smaller problems
- Understand the problem
  - what result is expected
  - what process can provide these results
  - what parameters are needed for these processes
  - write function descriptions in English telling what the function should do

Problem Solving Strategy

- C++ Syntax Typically Obscures Understanding
  - write out your solution on paper FIRST
  - use flow charts or pseudocode
  - translate to C++ syntax on paper
  - try not to compose code at a terminal
- Great Answers Don’t Come the First Time
  - iteratively refine and enhance partial solutions

Testing Strategy

- How Do You Test Functions?
  - Test One Function At A Time
  - Display Intermediate Results
  - You May Need To Create Test Data To Use Via “Driver Programs”
  - If The Function Being Tested Calls Other Functions, Create “Stubs”
  - Try Varying One Thing At A Time
    - if something goes wrong, you know what changed
Testing Strategy

• Drivers
  – allows you to test a function without all the rest of a program
  – just to execute the function and show its results
  – often, provides a loop to retest the function on different arguments

Testing Strategy

• Stubs
  – simplified version of a function not written or tested yet
  – often used when testing another function
  – does not necessarily deliver correct values
  – works best when stubs are replaced by actual functions, one at a time

Time For Our Next Demo!

• TestDriver.cpp

(See Handout For Example 2)
Summarizing Our Second Demo!

- Drivers Are Throwaway Code Meant To Exercise Other Code
- Stubs Are Fake StandIns For Code That Will Be Fleshed Out Later

An assert Macro

- Useful In Debugging
- Stops Execution So Problems Can Be Corrected

An assert Macro Example

- Given Function Declaration:
  ```
  void computeCoin(int coinValue, int& number, int& amountLeft);
  // Precondition: 0 < coinValue < 100
  // Postcondition: number set to max. number of coins
  ```
- Check precondition:
  ```
  assert(0 < coinValue) && (coinValue < 100)
  assert(0 <= amountLeft) && (amountLeft < 100));
  - If precondition not satisfied → condition is false → program execution terminates!
  ```
assert On/Off

- Preprocessor Provides This For Us To Use
  
  ```
  #define NDEBUG
  #include <cassert>
  ```

- Add "#define" line before #include line
  - Turns OFF all assertions throughout program

- Remove "#define" line (or comment out)
  - Turns assertions back on

Summary

- void Functions
- return Statement
- PostFix and Prefix ++, -- Operators
- Default Value Arguments
- Parameter Passing Mechanisms
- Problem Solving and Testing Strategy