APPENDIX A

This page is left blank intentionally

Managing I/O Formatting

Manipulators are the most common way to control output formatting.
To use I/O *manipulators* you must include iomanip.h file in your source code.

Default Floating-point Format

Unless you use I/O manipulators (or their equivalent), the default format for each floating-point number depends on its value.

- No decimal point: 1.0000 prints as 1
- No trailing zeros: 1.5000 prints as 1.5
- Scientific notation for large/small numbers: 1234567890.0 prints as 1.23457e+09

I/O Manipulators

The following manipulators control the output stream format. Include <iomanip.h> if you use any manipulators that have parameters. The Range column tells how long the manipulator will take effect: *now* inserts something at that point, *next* affects only the next data element, and *all* affects all subsequent data elements for the output stream.

Manip.	Range	Description
General		
endl	now	Write a newline ('\n') and flush buffer.
setw(*n*)	next	Sets minimum field width on output. This sets the minimum size of the field - a larger number will use more columns. Applies only to the next element inserted in the output. Uses left and right to justify the data appropriately in the field. Output is right justified by default. Equivalent to cout.width(*n*); To print a column of right justified numbers in a seven column field: `cout << setw(7) << n << endl;`
left	all	Left justifies output in field width. Only useful after `setw(n)`.
right	all	Right justifies output in field width. Since this is the default, it is only used to override the effects of `left`. Only useful after `setw(n)`.
setfill(*ch*)	all	Only useful after `setw`. If a value does not entirely fill a field, the character *ch* will be used to fill in the other characters. Default value is blank. Same effects as `cout.fill(ch)` For example, to print a number in a 4 character field with leading zeros (eg, 0007): `cout << setw(4) << setfill('0') << n << endl;`
For floating point values		
setprecision(*n*)	all	Sets the number of digits printed to the right of the decimal point. This applies to all subsequent floating point numbers written to that output stream. However, this won't make floating-point "integers" print with a decimal point. It's necessary to use `fixed` for that effect. Equivalent to cout.precision(*n*);
fixed	all	Used fixed-point notation for floating-point numbers. Opposite of `scientific`. If no precision has already been specified, it will set the precision to 6.
scientific	all	Formats floating-point numbers in scientific notation. Opposite of `fixed`.
For `bool` values		
Boolalpha noboolalpha	all	Uses alphabetic representation (`true` and `false`) for `bool` values. Turned off with `noboolalpha`.
Other		
		showpoint, noshowpoint, uppercase, nouppercase, dec, oct, hex, setbase(*8*|*10*|*16*), showbase, noshowbase, ends, showpos, noshowpos, skipws, noskipws, ws, internal, flush, unitbuf, nounitbuf, setiosflags(*f*), resetiosflags(f)

Example

```
const float tenth = 0.1;
  const float one   = 1.0;
  const float big   = 1234567890.0;

  cout << “A. “                    << tenth << “, “ << one << “, “ << big << endl;
  cout << “B. “ << fixed           << tenth << “, “ << one << “, “ << big << endl;
  cout << “C. “ << scientific      << tenth << “, “ << one << “, “ << big << endl;
cout << “D. “ << fixed << setprecision(3) << tenth << “, “ << one;
cout << “, “ << big << endl;
cout << “E. “ << setprecision(20) << tenth << endl;
cout << “F. “ << setw(8) << setfill(‘*’) << 34 << 45 << endl;
```

Output:

```
A. 0.1, 1, 1.23457e+09
B. 0.100000, 1.000000, 1234567936.000000
C. 1.000000e-01, 1.000000e+00, 1.234568e+09
D. 0.100, 1.000, , 1234567936.000
E. 0.10000000149011611938
F. ******3445
```

More I/O Manipulators

Function	Direction	Action
dec	in/out	Sets the base to decimal
oct	in/out	Sets the base to octal
hex	in/out	Sets the base to hexadecimal
ws	in	Extract white space characters
endl	out	Inserts a newline, flushes stream
ends	out	Inserts a nul byte
flush	out	Flushes a stream
setbase(int b)	in/out	Sets conversion base (0, 8, 10, 16). Base 0 means use base 10 for output, use C parsing rules for integer literals on input.
setiosflags(long f)	in/out	Set specified bits
resetiosflags(long f)	in/out	Clears specified bits
setfill(char c)	out	Sets the fill character
setprecision(int n)	out	Sets precision to n digits after the decimal point
setw(int w)	in/out	Sets the total field width

Character I/O Functions

Member Functions	Description
void eatwhite(void);	Extracts white space from the stream by advancing the get pointer past spaces and tabs.
int gcount(void);	Returns the number of characters extracted in the last extraction.
int get(void);	Extracts the next character form the input stream and returns it. An EOF (-1) is returned upon end of input.

istream &get(signed char &c);
istream &get(unsigned char &c);
	Extracts the next character from the input stream.
	Returns the input stream.

istream &get(signed char *s, int n, char t = ‘\n’); istream &get(unsigned char *s, int n, char t = ‘\n’);	Extracts up to n characters into s, Stopping when the termination character is found. The termination character is not extracted or stored in s. Returns the input stream.
istream &getline(signed char *s, int n, char t=’\n’); istream &getline(unsigned char *s, int n, char t = ‘\n’);	Extracts up to n characters into s, Stopping when the termination character is found. The termination character is extracted but not stored in s. Returns the input stream.
istream &ignore(int n, int t=EOF);	Extracts and discards up to n characters, or until the termination character is found. The termination character is removed from the input stream. The input stream is returned.
int peek(void);	Returns the next character from the input stream without extracting it. An EOF (-1) is returned upon end of input.
ostream &put(char c);	Inserts a character into the output stream. Returns the output stream.
istream &putback(char c);	Pushes back the character onto the input stream. The input stream is returned.

Functions for Setting and Testing the State of a Stream

Function	Description
int rdstate()	Returns current stream state
int good()	Returns nonzero if in good state
int eof()	Returns nonzero if at end of file
int fail()	Returns nonzero if **failbit**, **badbit**, or **hardfail** is set
int bad()	Returns nonzero if **badbit** or **hardfail** is set
void clear(int v=o);	Sets the state, (default is "good")
operator void*();	Returns 0 if **failbit**, **badbit**, or **hardfail** is set
in operator!();	Returns nonzero if **failbit**, **badbit**, or **hardfail** is set

Formatting Flags

Label	Value	Action
skipws	0x0001	Skip white space on input
left	0x0002	Left justify output
right	0x0004	Right justify ouput
internal	0x0008	Use padding after sign or base indicator
dec	0x0010	Use decimal conversion
oct	0x0020	Use octal conversion
hex	0x0040	Use hexadecimal conversion
showbase	0x0080	Use base indicator on output
showprint	0x100	Always show decimal point and trailing zeros on floating point output
uppercase	0x0200	Use uppercase for hex output
showpos	0x0400	Add '+' to positive integers on output
scientific	0x0800	Use exponential floating notation
fixed	0x1000	Use fixed point floating notation
unitbuf	0x2000	Flush all streams after output
stdio	0x4000	Flush cout, cerr after output

This page is left blank intentionally

APPENDIX B

This page is left blank intentionally

C++ Operators for Overloading

Operator	Description	Type	Associativity	Precedence
,	Comma Operator	binary	Left	1
=	Assignment Operator	binary	Right	2
+=	Assignment Operator	binary	Right	2
-=	Assignment Operator	binary	Right	2
*=	Assignment Operator	binary	Right	2
/=	Assignment Operator	binary	Right	2
|=	Assignment Operator	binary	Right	2
^=	Assignment Operator	binary	Right	2
&=	Assignment Operator	binary	Right	2
%=	Assignment Operator	binary	Right	2
<<=	Assignment Operator	binary	Right	2
>>=	Assignment Operator	binary	Right	2
||	Logical OR	binary	Left	4
&&	Logical AND	binary	Left	5
|	Bitwise OR	binary	Left	6
^	Bitwise XOR	binary	Left	7
&	Bitwise AND	binary	Left	8
==	Equality	binary	Left	9
!=	Inequality	binary	Left	9
<	Less Than	binary	Left	10
<=	Less than/equal	binary	Left	10
>	Greater than	binary	Left	10
>=	Greater than/equal	binary	Left	10
<<	Left shift	binary	Left	11
>>	Right shift	binary	Left	11
+	Addition	binary	Left	12
-	Subtraction	binary	Left	12
*	Multiplication	binary	Left	13
/	Division	binary	Left	13
%	Modulo	binary	Left	13
->*	Pointer to member	binary	Left	14
*	Pointer to member	binary	Left	14

C++ Operators for Overloading

Operator	Description	Type	Associativity	Precedence
++	Increment	unary	Right	15
--	Decrement	unary	Right	15
!	Logical NOT	unary	Right	15
~	Bitwise NOT	unary	Right	15
+	Unary plus	unary	Right	15
-	Unary minus	unary	Right	15
*	Pointer dereference	unary	Right	15
&	Address of	unary	Right	15
()	Typecast	binary	Right	15
new	Allocate	binary	Right	15
delete	De-allocate	binary	Right	15
->	Member selector	binary	Left	16
[]	Array index	binary	Left	16
()	Function call	binary	Left	16

This page is left blank intentionally

APPENDIX C

This page is left blank intentionally

Specification for Programming Projects

Please try to give your project a professional look. Turn-in Your program on the 1.44 MB floppy disk. Projects will be graded according to how well you follow the specifications given below.

Please include the following information on the first page of the source code of your project:

```
//////////////////////////////////////////
// Your Name:
// Project Number:
// Due Date:
// Compiler Used:
// Compiler Version:
// File Name:
//////////////////////////////////////////
```

Programming Conventions

All keywords and variable names must be in lower case. Variables must have meaningful, self-descriptive names. All #define and constant variable names are to be in UPPER CASE. Only one statement per line should be used. Indentation must be used for the body of loops and decisions. Blank lines should be used for separation of code to make the program readable. All functions should be easily identifiable, i.e. separated from each other by a header such as:

```
//////////////////////////
// Function: main
// Purpose:
// Parameters:
// Local Variables:
//   pdollar: Holds amount to purchase office equipment.
//   pitem : Holds office equipment item purchased.
//////////////////////////
```

Describe the purpose of the function. Functions should do just one task. In-line comments should be used when necessary.

Format For Class Header

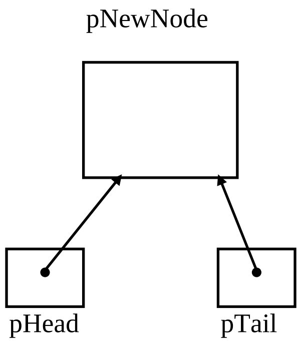
```
/////////////////////////////////////////////////////////////
// Class Name:
// Purpose:
//
//
//
// Inline Functions:
//  Function Name:
//  Functions Prototype:
//  Description:
//
// Other Member Functions:
//  Function Name:
//  Functions Prototype:
//
// Friend Function:
//  Function Name:
//  Functions Prototype:
//  Description:
//
///////////////////////////////////////////////////////////
```

This page is left blank intentionally

Insertion Methods for Doubly Link List

Case 1

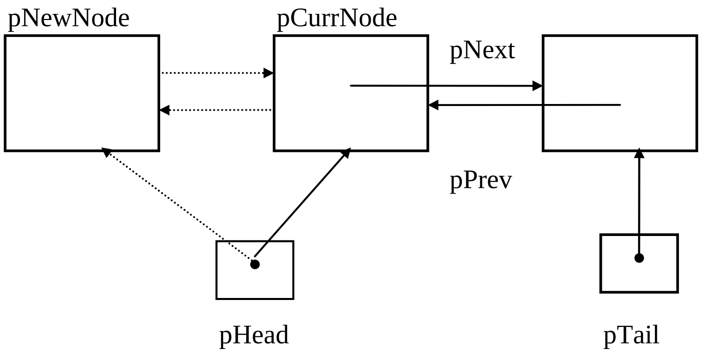
1 record only



```
PHead = pNewNode;
pTail = pNewNode;
```

Case 2

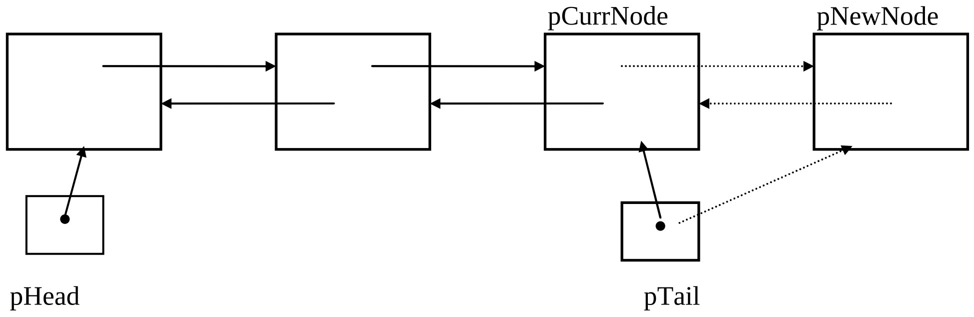
Inserting at the beginning of link list:



```
PNewNode->pNext = pCurrNode;
pNewNode->pPrev = 0;
pHead = pNewNode;
pCurrNode->pPrev = pNewNode;
```

Case 3

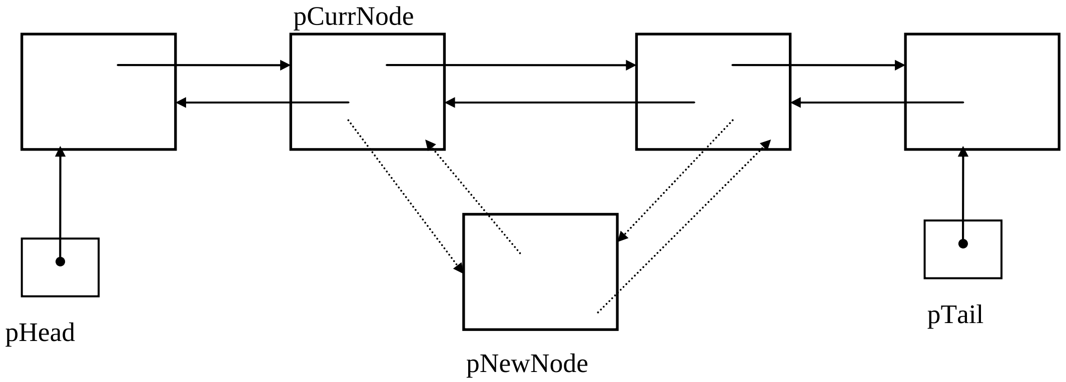
Inserting at the end of Link List:



```
pNewNode->pNext = 0;
PNewNode->pPrev = pCurrNode;
pCurrNode->pNext = pNewNode;
pTail = pNewNode;
```

Case 4

Inserting in between Nodes:



```
PNewNode->pPrev = pCurrNode;
pNewNode->pNext = pCurrNode->pNext;
pCurrNode->pNext = pNewNode;
pNewNode->pNext->pPrev = pNewNode;
```

This page is left blank intentionally

APPENDIX D

This page is left blank intentionally

UCSC Extension Quiz # 1

Instructor: Suleman Saya Name:____________________

1. State which of the following are true and which are false:

□ a) Members of a class specified as private are accessible anywhere an object of the class is in scope.

□ b) Static class variables have class scope.

□ c) A function declared static cannot access static class members.

□ d) It is ok to have a return statement in a constructor.

□ e) A constructor can take a pass by value argument of its own class type.

□ f) It is ok to overload the destructor.

□ g) For default arguments, left most arguments can be unspecified when passing the parameter.

□ h) A reference operator must be initialized at declaration and cannot be changed.

□ i) Assignment operator causes the call to the copy constructor.

□ j) It is ok to return a reference to a local variable of a function.

2. Fill in the blanks in each of the following: (30 Points)

a) For inline functions _______ is inserted at the location of the call and appropriate variables are renamed.

b) Inline function has the same ________ and arguments passing semantics as standard functions.

c) You can not assign _______ to a this pointer.

d) ______ member function does not have this pointer.

e) A function declared static cannot access _________ class members.

f) _______ class member have class scope.

g) Members of a class specified as _______ are accessible anywhere an object of the class is in scope.

h) ________ and _______ cannot be declared as constant member functions.

i) All static data members are initialized to _______ by default.

j) A reference is an _______ for an actual variable.

k) Constructor can not have a _______ statement.

l) C++ uses _______ and _______ operators for memory management.

m) The _______ makes the variable or object of any type read-only.

n) _______ data member exist before any object is created.

o) _______ pointer is passed implicitly to every non-static member function.

3. Select the correct output for the following four programs:

```
a.  #include <iostream.h>
class CNumber {
        public:
                CNumber (int Hex) { Integer =  Hex; }
                void Add() {++Integer; }
                void Show() const {cout << Integer << "\n"; }
        private:
                int Integer;
};

void main()
{
        CNumber Alpha(31);
        const CNumber Beta =  23;

        Alpha.Add();
        Alpha.Show();
        Beta.Show();
}
```

Select the correct output:

1. 31 23
2. 32 23
3. Non of the above.

```
b.  #include <iostream.h>
    class CNumber {
       public:
       CNumber(int Hex) { Integer = Hex; }
       void add() { ++Integer; }
       void show() const {cout << Integer << " "; }
       private:
       int Integer;
    };

    void main()
    {
       CNumber alpha(60);
       const CNumber beta = 58;

       alpha.add();
       alpha.show();
       beta.show();
    }
```

Select the correct output:

1. 60 58
2. 61 58
3. Non of the above.

****

```
c.  #include <iostream.h>
class CFruit {
    public:
        CFruit(int n) { m_nCount = n++; }
    private:
        int m_nCount;
        int Pick() { return (m_nCount); }
};

void main()
{
    CFruit Apples(120);
    cout << "Value using object: " << Apples.Pick() << '\n';
}
```

Select the correct output:

1. Value using object: 121
2. Value using object: 120
3. Error In the program.

d.

```
#include <iostream.h>
class CCountDown {
    public:
        void Set(int n) { m_nItem = --n; }

        void Show() { cout << m_nItem << "\n"; }
    private:
        static int m_nItem;
};

int CCountDown::m_nItem;
void main()
{
    CCountDown Obj1, Obj2;

    Obj1.Set(20);
    Obj1.Show();
    Obj2.Show();
}
```

Select the correct output:

1. 19 19
2. 19 20
3. Error In the program

```
e.  #include <iostream.h>
class CStudent {
        int nID;
        CStudent() { nID = 100; }
public:
        void ShowID();
};

void CStudent::ShowID()
{
        cout << ++this->nID << "\n";
}

void main()
{
        CStudent Obj;
        Obj.ShowID();
}
```

Select the correct output:

1. 100
2. 101
3. Error In the program