C++ Programming, Comprehensive

Jagadeesh Vasudevamurthy Ph.D
Instructor
UCSC Extension in SiliconValley
Santa Clara, CA 95054 USA
jvasudev@ucsc.edu

CMPR.X404 - C++ Programming, Comprehensive

Dates: Sep 26, 2019 - Dec 5, 2019 Time: 6:00 PM PDT

Location
SANTA CLARA

Instructors
Jagadeesh Vasudevamurthy

Draft 16
Sept 2019

Recommended Texts:

C++ Primer (5th edition), Stanley Lippman and Josee Lajoie, Addison-Wesley Professional, ISBN-10: 0321714113 ISBN-13: 978-0321714114

VASUDEVAMURTHY

नहि ज्ञानेन सदृशं

Nahi Jnanena Sadrusham

Nothing is equivalent to knowledge

This work is dedicated to my mother V.Radhabai
who sacrificed her life to provide me the best possible education.

VASUDEVAMURTHY

Contents

1 Introduction **9**

1.1 Introduction 9

1.2 Installing Microsoft Visual C++ 2015 compiler 11

1.3 Writing, compiling and debugging first program using Microsoft Visual C++ 2015 compiler 17

1.4 First C++ program revisted 26

1.4.1 *util.h* 26

1.4.2 *class* helloworld 28

1.4.3 **C++**code for *class* helloworld 30

1.5 Basic data types 34

1.5.1 Boolean number system 34

1.5.2 **bool** data type 36

1.5.3 **char** data type 36

1.5.4 **int** data type 38

1.5.5 **float and double** data type 41

1.5.6 **C++**code for illustrating basic data types 41

1.6 User defined data types 47

1.7 **auto** 49

1.8 Initializer list assignment 60

1.9 Arithmetic operations 70

1.9.1 **C++**code for illustrating arithmetic operations 71

1.10 Relational operations 77

VASUDEVAMURTHY

1.10.1 C++code for illustrating relational operations 78
1.11 Logical operations 82
1.11.1 C++code for illustrating logical operations 82
1.12 Bitwise operations 86
1.12.1 C++code for illustrating bitwise operations 88
1.13 Assignment operations 93
1.14 Control statements 93
1.14.1 C++code for illustrating control statements 96
1.15 Enumerations(enum) 108
1.15.1 C++code for illustrating enumeration 111
1.16 Introductions to functions 116
1.16.1 C++code for illustrating call by value function 116
1.17 Introductions to pointers 121
1.17.1 C++code for illustrating pointers 124
1.18 Understanding **const** 129
1.19 Introductions to arrays 132
1.19.1 C++code for illustrating array and pointers to array elements 134
1.20 Basic stream output and stream input 140
1.20.1 C++code for illustrating stream output 142
1.20.2 C++code for illustrating stream input 148
1.21 Problem set 154

2 C Strings 175
2.1 Introduction 175
2.2 Character manpulation 175
2.3 C Strings 182
2.4 Problem set 189

3 Concept of reference 191
3.1 Introduction 191

VASUDEVAMURTHY

3.2 Concept of swapping using reference 193
3.3 FAQ on references 195
3.4 Passing char* by value, address and reference 203

4 Dynamic memory allocation 205
4.1 Introduction 205
4.2 Static and dynamic memory allocation 205
4.3 Allocating two dimensional array 211
4.3.1 Static allocation 211
4.3.2 Dynamic allocation 211
4.4 *unique_ptr* 220
4.5 Problem set 230

5 Function overloading and default function arguments 239
5.1 Introduction 239
5.2 Function overloading 239
5.3 Default function arguments 241

6 C++ class 243
6.1 Introduction 243
6.2 Need for *class* 243
6.3 *class* and objects 247
6.4 Accessing class members using dot and arrow operator 247
6.5 Accessing private class members 251
6.6 Separating interface and implementation 253
6.7 Allocating class objects on stack and heaps, phase 1 258
6.8 Shallow copying of objects, phase 1 267
6.9 Creating temporary objects on the stack, phase1 270
6.10 inline function 273
6.11 Understanding **const** 277
6.12 Need for constructors 280

VASUDEVAMURTHY

6.13 constructors = default . . . 287
6.14 Creating temporary objects on the stack, phase 2 . . . 290
6.15 Shallow copying of objects, phase 2 . . . 293
6.16 Copy constructor and equal operator . . . 296
6.16.1 Fraction class . . . 308
6.17 Private constructors . . . 312
6.18 Constructor initialization list . . . 314
6.18.1 Initialization versus assignment . . . 314
6.18.2 Order of initialization . . . 314
6.19 Static member of a class . . . 319
6.20 Size of a class . . . 319
6.21 Need for friend . . . 319
6.21.1 **friend** function . . . 319
6.21.2 **friend** class . . . 329
6.22 Type conversions . . . 332
6.23 Usage of class string . . . 339
6.24 Object composition . . . 339
6.25 Returning member variables of a class . . . 346
6.26 Problem set . . . 349

7 Operator overloading **365**
7.1 Introduction . . . 365
7.2 Overloading an operator as a member function or as a friend . . . 365
7.3 Operators supported by C++ on basic types . . . 377
7.4 Operators that cannot be overloaded . . . 380
7.5 Operators precedence in C++ . . . 380
7.6 Int class . . . 382
7.6.1 Unary operators . . . 384
7.6.2 Binary operators . . . 390
7.6.3 Implementing **Int** class . . . 393

VASUDEVAMURTHY

7.7 A set class 399
7.8 A long number class 411
7.9 mstring class 412

8 Inheritance 421
8.1 Introduction 421
8.2 Why Inheritance? 421
8.3 Need for keyword: protected 423
8.4 Overriding or hiding base class member functions 428
8.5 Writing constructor for a derived class 432
8.6 Writing destructor for a derived class 432
8.7 Writing copy constructor and equal operator for a derived class in a BAD way 434
8.8 Writing copy constructor and equal operator for a derived class in a CORRECT way 434
8.9 Writing copy constructor for a derived class 440
8.10 Writing equal operator for a derived class 440
8.11 Writing insertion operator for a derived class 442
8.12 Using base and derived class objects 442
8.13 Problems of static binding and need for polymorphism 453
8.14 Polymorphism through keyword virtual 459
8.15 Rewriting exam program using polymorphism 463
8.16 Abstract class and concept of pure virtual function 472
8.17 Casting from derived to base and base to derived 476
8.18 Implementing polymorphism without using **virtual** 481
8.19 Problem set 485

9 Namespaces 495
9.1 Introduction 495
9.2 What is the problem? 495
9.3 Need for namespace 496

VASUDEVAMURTHY

10 Template **503**
10.1 Introduction 503
10.2 Macros 503
10.3 Function templates 507
10.4 Class templates 512
10.5 Integer stack implementation 512
10.6 Generic stack implementation using typedef 519
10.7 Stack implementation using inheritence 527
10.7.1 Problem of using objects and static binding 527
10.8 Stack implementation using template 542

11 Exception handling **555**
11.1 Introduction 555
11.2 Problem 555
11.3 Try and catch 559
11.4 Stack example 564

12 Standard Template Library **575**
12.1 Introduction 575
12.2 **util** class 575
12.3 **Concept of l-value and r-value** 579
12.4 **Int** class 585
12.5 **vector** class 597
12.6 **string** class 614
12.7 **deque** class 620
12.8 Problem set 629

VASUDEVAMURTHY

Chapter 1

Introduction

1.1 Introduction

VASUDEVAMURTHY

Figure 1.1: Typical C++ environment

VASUDEVAMURTHY

1.2 Installing Microsoft Visual C++ 2015 compiler

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Getting free Microsoft Visual Studio Community 2015

Jagadeesh Vasudevamurthy

06/22/2016

1. Go this website:

https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx

Click on Download

VASUDEVAMURTHY

Visual Studio

Community 2015
with Updates

Choose your installation location

C:\Program Files (x86)\Microsoft Visual Studio 14.0 ...

Setup requires up to 18 GB across all drives.

By clicking the "Next" button, I acknowledge that I accept the License Terms and Privacy Statement.

Cancel Next

Visual Studio
Community 2015
with Updates
Select features
Programming Languages
Visual C++
Common Tools for Visual C++ 2015
Microsoft Foundation Classes for C++
Windows XP Support for C++
Visual F#
Python Tools for Visual Studio (March 2016)
Windows and Web Development
ClickOnce Publishing Tools
Microsoft SQL Server Data Tools
Microsoft Web Developer Tools
Select All
Reset Defaults
Setup requires up to 16 GB across all drives.
Back
Next

Visual Studio
Community 2015
with Updates
Selected features
MICROSOFT SOFTWARE
C#/.NET (Xamarin v4.1.0)
License Terms
C#/.NET (Xamarin v4.0.2 or earlier) will be removed
License Terms
Windows 8.1 SDK and Universal CRT SDK
Common Tools for Visual C++ 2015
Tools for Universal Windows Apps (1.3.2) and Windows 10 SDK (10.0.10586)
THIRD-PARTY SOFTWARE
Android Native Development Kit (R10E, 32 bits)
License Terms
Android SDK Setup (API Level 23)
License Terms
Setup requires up to 16 GB across all drives.
By clicking "Install" you agree to the license terms of the software and components you have selected to install. You are responsible for reading and accepting these license terms. Microsoft grants you no rights for any third party software you have selected; it is provided by the third parties indicated in the applicable license terms.
Back
Install

With this installation is you should see this on your desktop
Make a short cut on desktop

VASUDEVAMURTHY

1.3 Writing, compiling and debugging first program using Microsoft Visual C++ 2015 compiler

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Writing, compiling and debugging first program

Jagadeesh Vasudevamurthy
03/21/2016

1. Double click on this icon

2. Create a new project

3. Click on New Project

VASUDEVAMURTHY

1. **Select EMPTY PROJECT**
2. **Name: simple2015**
3. **Location: C:\jag\msft\2015**
4. **Click OK**

4. You will then see a screen as follows:

5.

1. **Right click on Source file**
2. **select Add -> New Item as follows**

6.

1.Select C++ file (cpp)
2. Name: test.cpp
3. Location: C:\test
4. Click on Add

7 Now you will a screen as follows:

8 Enter your program as below and save the file with Ctrl S together

9. Compiling the above program by pressing F7

If everything is right, you should get this

```
1>------ Build started: Project: simple2013, Configuration: Debug Win32 ------
1>  test.cpp
1>  simple2013.vcxproj -> c:\jag\msft\simple2013\simple2013\Debug\simple2013.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

10. Setting a break point

Go to last line }
Left click on }
Press F9
You will see a red dot as follows

11. Running the program

Press F5

You will see output as follows:

VASUDEVAMURTHY

Compiling using gnu g++ compiler on Linux system

```
Program: test.cpp

#include <iostream>
using namespace std;

int main() {
        cout << "Welcome to CE589B course\n" ;
        return 0 ;
}
```

```
Compiling using g++
g++ test.cpp -o test
```

```
Running the program
./test

Welcome to CE589B course
```

VASUDEVAMURTHY

1.4 First C++ program revisted

```
1 //file: 1.cpp                    //Comment for humans. Ignored by compiler
2 #include <iostream>          //Tells the preprocessor to include this file that has I/O operations
3 using namespace std;          //Use cout object that is defined in std namespace
4 int main() {                  Begin of main
5     cout << "Hello World" << endl ;  //Send the message "Hello World" to the object cout, so that it will be written on your screen
6     return 0 ;                After you write "Hello World", enter a line feed
7 }
8                               Name of the procedure that will called first by the compiler
End of main

This procedure returns an integer at the end so that the caller can use it.
0 means the procedure exited successfully
1 means failure
```

Note ; after every line of code except #include

```
1 //file: 1.cpp
2 #include <iostream>
3 //using namespace std;
4 int main() {
5     cout << "Hello World" << endl ;
6     return 0 ;
7 }
8
```

Compiler will no longer understand cout and endl

```
1.cpp
..\..\..\jag\c++\course\code\ch1\1.cpp(5) : error C2065: 'cout' : undeclared identifier
..\..\..\jag\c++\course\code\ch1\1.cpp(5) : error C2065: 'endl' : undeclared identifier
```

Figure 1.2: Understanding first C++ program

1.4.1 *util.h*

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: util.h
----------------------------------------------------------------*/

/*----------------------------------------------------------------
include this file for all the programs
----------------------------------------------------------------*/
#ifndef UTIL_H
#define UTIL_H

/*----------------------------------------------------------------
Basic include files
----------------------------------------------------------------*/
#include <iostream>
#include <fstream>

#include <iomanip>      // std::setprecision
using namespace std;

#ifdef _WIN32
#include <cassert>
#include <ctime>
#else
#include <assert.h>
#include <time.h>
#include <string.h> //for strlen,strcat and strcpy on linux
#endif

//'sprintf' : This function or variable may be unsafe.Consider using sprintf_s instead.To disable deprecation,
//use _CRT_SECURE_NO_WARNINGS
//To overcome above warning
#ifdef _MSC_VER
#pragma warning(disable: 4996) /* Disable deprecation */
#endif


/*----------------------------------------------------------------
STL
----------------------------------------------------------------*/
#include <stdexcept> //Without this catch will NOT work on Linux
#include <vector>
#include <string>
#include <unordered_set>
#include <set>
#include <map>
#include <queue>
#include <algorithm>
```

VASUDEVAMURTHY

util.h

```
#ifndef UTIL_H
#define UTIL_H
    #include <iostream>
    #ifdef WIN32
    #include <fstream>
    #include <cassert>
    #include <ctime>
    #include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
    #else
    #include <fstream>
    #include <assert.h>
    #include <time.h>
    #endif
    #include <iomanip>     // std::setprecision
    using namespace std;
#endif
//'sprintf' : This function or variable may be unsafe.Consider using sprintf_s instead.To disable deprecation,
//use _CRT_SECURE_NO_WARNINGS
//To overcome above warning

#ifdef _MSC_VER
#pragma warning(disable: 4996) /* Disable deprecation */
#endif
```

Figure 1.3: *util.h*

1.4.2 *class* helloworld

VASUDEVAMURTHY

Figure 1.4: *class* helloworld

VASUDEVAMURTHY

1.4.3 C++code for *class* helloworld

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: helloword.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has helloworld class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef HELLOWORLD_H
#define HELLOWORLD_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of helloworld class
-----------------------------------------------------------------*/
class helloworld{
  public:
    void welcome() ;
  private:

};

#endif


```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: helloword.cpp
-------------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has helloworld class definition
-------------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-------------------------------------------------------------------*/
#include "helloworld.h"

/*-----------------------------------------------------------------
Definition of welcome routine of helloworld class
-------------------------------------------------------------------*/
void helloworld::welcome() {
  cout << "Welcome to CS464" << endl ;
}

//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: helloworldtest.cpp

On linux:
g++ helloworld.cpp helloworldtest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test helloworld object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "helloworld.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  helloworld a ;
  a.welcome() ;
  helloworld b ;
  b.welcome() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF
```

VASUDEVAMURTHY

1.5 Basic data types

1.5.1 Boolean number system

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Figure 1.5: Boolean numbers

VASUDEVAMURTHY

1.5.2 bool data type

Figure 1.6: **bool** data type

1.5.3 char data type

VASUDEVAMURTHY

Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH	(start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX	(start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX	(end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT	(end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ	(enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK	(acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL	(bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS	(backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB	(horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT	(vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR	(carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO	(shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI	(shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE	(data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1	(device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2	(device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3	(device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4	(device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK	(negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN	(synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB	(end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN	(cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM	(end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB	(substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC	(escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS	(file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS	(group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS	(record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US	(unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Figure 1.7: **char** data type

VASUDEVAMURTHY

1.5.4 int data type

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Figure 1.8: Maximum and minimum number that can be represented in an integer

VASUDEVAMURTHY

```
void datatype::test_int() {
  cout << "sizeof(unsigned short int) = " << sizeof(unsigned short int) <<  " bytes" <<  endl;
  cout << "sizeof(unsigned int) = " << sizeof(unsigned int) <<  " bytes" <<  endl;
  cout << "sizeof(unsigned long int) = " << sizeof(unsigned long int) <<  " bytes" <<  endl;
  cout << "sizeof(short int) = " << sizeof(short int) <<  " bytes" <<  endl;
  cout << "sizeof(int) = " << sizeof(int) <<  " bytes" <<  endl;
  cout << "sizeof(long int) = " << sizeof(long int) <<  " bytes" <<  endl;

  unsigned short short_unsigned_max = 0xFFFF ;
  cout << "short_unsigned_max= " <<  short_unsigned_max <<  endl ;
  unsigned short short_unsigned_min = 0 ;
  cout << "short_unsigned_min = " <<   short_unsigned_min <<  endl ;

  short int short_int_max = 0x7FFF ;
  cout << "short_int_max = " <<  short_int_max <<  endl ;
  short int short_int_min = 0x8000 ;
  cout << "short_int_min = " <<  short_int_min <<  endl ;

  unsigned int unsigned_int_max = 0xFFFFFFFF ;
  cout << "unsigned_int_max= " <<  unsigned_int_max <<  endl ;
  unsigned int unsigned_int_min = 0 ;
  cout << "unsigned_int_min = " <<   unsigned_int_min <<  endl ;

  int int_max = 0x7FFFFFFF ;
  cout << "int_max = " <<  int_max <<  endl ;
  int int_min = 0x80000000 ;
  cout << "int_max = " <<  int_min <<  endl ;

  int i =  2147483648 ;
  cout << "int i = " <<  i  <<  endl ;

  unsigned long unsigned_long_max = 0xFFFFFFFF ;
  cout << "unsigned_long_max= " <<  unsigned_long_max <<  endl ;
  unsigned long unsigned_long_min = 0 ;
  cout << "unsigned_long_min = " <<   unsigned_long_min <<  endl ;

  long long_max = 0x7FFFFFFF ;
  cout << "long_max = " <<  long_max <<  endl ;
  long long_min = 0x80000000 ;
  cout << "long_min = " <<  long_min <<  endl ;

  long long max = 0x7FFFFFFF;
  unsigned long long umax = 0xFFFFFFFF;
}
```

```
sizeof(unsigned short int) = 2 bytes
sizeof(unsigned int) = 4 bytes
sizeof(unsigned long int) = 4 bytes
sizeof(short int) = 2 bytes
sizeof(int) = 4 bytes
sizeof(long int) = 4 bytes
```

```
short_unsigned_max= 65535
short_unsigned_min = 0
```

```
short_int_max = 32767
short_int_min = -32768
```

```
unsigned_int_max= 4294967295
unsigned_int_min = 0
```

```
int_max = 2147483647
int_max = -2147483648
```

```
int i = -2147483648
```

```
unsigned_long_max= 4294967295
unsigned_long_min = 0
```

```
long_max = 2147483647
long_min = -2147483648
```

```
long long max = 2147483647
long long min = -2147483648
unsigned long long max = 4294967295
```

Figure 1.9: **int** data type

VASUDEVAMURTHY

1.5.5 float and double data type

```
void datatype::test_real() {
  cout << "sizeof(float) = " << sizeof(float) <<  " bytes" <<  endl;
  float f = 67.8f;
  cout << "float f = " << f <<  endl ;


  cout << "sizeof(double) = " << sizeof(double) <<  " bytes" <<   endl;
  double d = 28789.789 ;
  cout << "double d = " << d << endl ;
  cout << "double d = " << std::setprecision(9) << d << endl ;
}
```

Figure 1.10: **float and double** data type

1.5.6 C++code for illustrating basic data types

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: datatype.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has datatype class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef DATATYPE_H
#define DATATYPE_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of datatype class
-----------------------------------------------------------------*/
class datatype{
public:
  void test_bool() ;
  void test_char() ;
  void test_int() ;
  void test_real() ;
  void test_long_long(); //C++11
private:

};

#endif


```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: datatype.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has datatype class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "datatype.h"

/*-----------------------------------------------------------------
Definition of routines of datatype class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
sizeof(bool) = 1 byte
bool b =  1
bool b =  0
bool b =  1
bool b =  1
bool b =  0
-----------------------------------------------------------------*/
void datatype::test_bool() {
  cout << "sizeof(bool) = " << sizeof(bool) <<  " byte" << endl;
  bool b = true ;
  cout << "bool b =  " << b << endl ;
  b = false ;
  cout << "bool b =  " << b << endl ;
  b = 256; //integer converted to bool. See warning
  //To prove any number other than 0 is true(1)
  cout << "bool b =  " << b << endl ;
  b = -127 ;
  cout << "bool b =  " << b << endl ;
  b =  0 ;
  cout << "bool b =  " << b << endl ;
  b = -127 ;
  cout << "bool b =  " << b << endl ;
}

/*-----------------------------------------------------------------
sizeof(char) = 1 byte
char c = a
char c = a
char c =
char c = \
char c =

char c = /
char c =
char c = a
-----------------------------------------------------------------*/
void datatype::test_char() {
  char c = 'g';
  cout << "sizeof(char) = " << sizeof(char) <<  " byte" << endl;
  c = 'a' ;
  cout << "char c = " <<  c <<  endl ;
  c = 97 ;
  cout << "char c = " <<  c <<  endl ;
  c = '\t' ;
  cout << "char c = " <<  c <<  endl ;
  c = '\\' ;
  cout << "char c = " <<  c <<  endl ;
```

VASUDEVAMURTHY

```
  c = '\n' ;
  cout << "char c = " <<  c <<  endl ;
  c = '/' ;
  cout << "char c = " <<  c <<  endl ;
  c = '\0' ;
  cout << "char c = " <<  c <<  endl ;
  c =  97+256  ;
  cout << "char c = " <<  c <<  endl ;
}

/*-----------------------------------------------------------------
sizeof(unsigned short int) = 2 bytes
sizeof(unsigned int) = 4 bytes
sizeof(unsigned long int) = 4 bytes
sizeof(short int) = 2 bytes
sizeof(int) = 4 bytes
sizeof(long int) = 4 bytes
short_unsigned_max= 65535
short_unsigned_min = 0
short_int_max = 32767
short_int_min = -32768
unsigned_int_max= 4294967295
unsigned_int_min = 0
int_max = 2147483647
int_max = -2147483648
int i = -2147483648
unsigned_long_max= 4294967295
unsigned_long_min = 0
long_max = 2147483647
long_min = -2147483648
-----------------------------------------------------------------*/
void datatype::test_int() {
  cout << "sizeof(unsigned short int) = " << sizeof(unsigned short int) <<  " bytes" <<  endl;
  cout << "sizeof(unsigned int) = " << sizeof(unsigned int) <<  " bytes" <<  endl;
  cout << "sizeof(unsigned long int) = " << sizeof(unsigned long int) <<  " bytes" <<  endl;
  cout << "sizeof(short int) = " << sizeof(short int) <<  " bytes" <<  endl;
  cout << "sizeof(int) = " << sizeof(int) <<  " bytes" <<  endl;
  cout << "sizeof(long int) = " << sizeof(long int) <<  " bytes" <<  endl;

  unsigned short short_unsigned_max = 0xFFFF ;
  cout << "short_unsigned_max= " <<  short_unsigned_max <<  endl ;
  unsigned short short_unsigned_min = 0 ;
  cout << "short_unsigned_min = " <<   short_unsigned_min <<  endl ;

  short int short_int_max = 0x7FFF ;
  cout << "short_int_max = " <<  short_int_max <<  endl ;
  short int short_int_min = 0x8000 ;
  cout << "short_int_min = " <<  short_int_min <<  endl ;

  unsigned int unsigned_int_max = 0xFFFFFFFF ;
  cout << "unsigned_int_max= " <<  unsigned_int_max <<  endl ;
  unsigned int unsigned_int_min = 0 ;
  cout << "unsigned_int_min = " <<   unsigned_int_min <<  endl ;

  int int_max = 0x7FFFFFFF ;
  cout << "int_max = " <<  int_max <<  endl ;
  int int_min = 0x80000000 ;
  cout << "int_max = " <<  int_min <<  endl ;

  int i =  2147483648 ;
  cout << "int i = " <<  i  <<  endl ;

  unsigned long unsigned_long_max = 0xFFFFFFFF ;
  cout << "unsigned_long_max= " <<  unsigned_long_max <<  endl ;
  unsigned long unsigned_long_min = 0 ;
  cout << "unsigned_long_min = " <<   unsigned_long_min <<  endl ;
```

```cpp

  long long_max = 0x7FFFFFFF ;
  cout << "long_max = " <<  long_max <<  endl ;
  long long_min = 0x80000000 ;
  cout << "long_min = " <<  long_min <<  endl ;
}

/*-----------------------------------------------------------------
sizeof(float) = 4 bytes
float f = 67.8
sizeof(double) = 8 bytes
double d = 28789.8
double d = 28789.789
-----------------------------------------------------------------*/
void datatype::test_real() {
  cout << "sizeof(float) = " << sizeof(float) <<   " bytes" <<  endl;
  float f = 67.8f;
  cout << "float f = " << f <<  endl ;

  cout << "sizeof(double) = " << sizeof(double) <<  " bytes" <<   endl;
  double d = 28789.789 ;
  cout << "double d = " << d << endl ;
  cout << "double d = " << std::setprecision(9) << d << endl ;
}

/*-----------------------------------------------------------------
sizeof(long long) = 8 bytes
sizeof(unsigned long long) = 8 bytes
unsigned long long is the same as unsigned long long int.
Its size is platform-dependent, but guaranteed
to be at least 64 bits
-----------------------------------------------------------------*/
void datatype::test_long_long(){
  cout << "sizeof(long long) = " << sizeof(long long) << " bytes" << endl;
  cout << "sizeof(unsigned long long) = " << sizeof(unsigned long long) << " bytes" << endl;
  long long x = 100000000000;
  cout << "x = " << x << endl;
  long long y = 100000000000LL;
  cout << "y = " << y << endl;
  //long long z = 0x7FFFFFFF;
  long long z = 0x7FFFFFFFFFFFFFFF;
  cout << "z = " << z << endl;
  long long z1 = z + 1;
  cout << "z1 = " << z1 << endl;
  unsigned long long uz = 0xFFFFFFFFFFFFFFFF;
  cout << "uz = " << uz << endl;
  unsigned long long uz1 = uz + 1;
  cout << "uz1 = " << uz1 << endl;
  unsigned long long uz2 = uz + 5;
  cout << "uz1 = " << uz2 << endl;
}

//EOF

```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: datatypetest.cpp

On linux:
g++ datatype.cpp datatypetest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test datatype object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "datatype.h"

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  datatype a ;
  a.test_bool() ;
  a.test_char() ;
  a.test_int() ;
  a.test_real() ;
  a.test_long_long();
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

1.6 User defined data types

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```
/*----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: uobject.cpp
----------------------------------------------------------------*/
#include <iostream>
using namespace std;
//'sprintf' : This function or variable may be unsafe.Consider using sprintf_s instead.To disable deprecation,
//use _CRT_SECURE_NO_WARNINGS
//To overcome above warning

#ifdef _MSC_VER
#pragma warning(disable: 4996) /* Disable deprecation */
#endif

struct book {
    char name[20] ;
    int  cost ;
    bool in ;
};

/*----------------------------------------------------------------
b1: Name = Algorithms Cost = 23 In
----------------------------------------------------------------*/
int main() {
    book b1;
    strcpy(b1.name,"Algorithms" );
    b1.cost = 23;
    b1.in = true ;
    cout << "b1: Name = " << b1.name << " Cost = " << b1.cost ;
    (b1.in) ? cout <<" In " : cout <<  " Out " ;
    cout << endl ;
    return 0 ;
}

```

VASUDEVAMURTHY

1.7 auto

auto

```
{
    bool b = true;
    char c = 'a';
    unsigned int n = 16;
    int m = 16;
    cout << "size of int " << sizeof(int) << endl;
    long l = 89890l;
    cout << "size of long " << sizeof(long) << endl;
    long long h = 12345678ll;
    cout << "size of long long " << sizeof(long long) << endl;
    float f = 7.8f;
    double d = 3.141456;
    cout << "size of double " << sizeof(double) << endl;
  }
{
    //To show you don't require data type at all
    auto b = true;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    auto c = 'a';
    cout << "type of c is " << typeid(c).name() << " Size = " << sizeof(c) << " Bytes. Value = " << c << endl;
    auto n = 16u; //unsigned int
    cout << "type of n is " << typeid(n).name() << " Size = " << sizeof(n) << " Bytes. Value = " << n << endl;
    auto m = 16;//signed int
    cout << "type of m is " << typeid(m).name() << " Size = " << sizeof(m) << " Bytes. Value = " << m << endl;
    auto l = 89890l;//signed long
    cout << "type of l is " << typeid(l).name() << " Size = " << sizeof(l) << " Bytes. Value = " << l << endl;
    auto h = 12345678ll; //signed long long
    cout << "type of h is " << typeid(h).name() << " Size = " << sizeof(h) << " Bytes. Value = " << h << endl;
    auto u = 12345678ull; //unsigned long long
    cout << "type of u is " << typeid(u).name() << " Size = " << sizeof(u) << " Bytes. Value = " << u << endl;
    auto f = 7.8f;
    cout << "type of f is " << typeid(f).name() << " Size = " << sizeof(f) << " Bytes. Value = " << f << endl;
    auto d = 3.141456;
    cout << "type of d is " << typeid(d).name() << " Size = " << sizeof(d) << " Bytes. Value = " << d << endl;
  }
```

Figure 1.11: **auto**

VASUDEVAMURTHY

auto

```
{
    //To show you cannot change data type when once you declared
    //Opposite to Python
    auto b = 3;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    //warning C4244: '=': conversion from 'double' to 'int', possible loss of data
    b = 3.141456;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    b = 5.141456;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    //Note: There is no context in which this conversion is possible
    //b = "UCSC";
    //cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
```

```
type of b is int Size = 4 Bytes. Value = 3
type of b is int Size = 4 Bytes. Value = 3
type of b is int Size = 4 Bytes. Value = 5
```

Figure 1.12: **auto**

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: learnauto.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has learnauto class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef learnauto_H
#define learnauto_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of learnauto class
-----------------------------------------------------------------*/
class learnauto {
public:
  void basicDataType();
  void testReturnAuto();
  void testSwap();
  void arrayInitialization();
  void objects();
private:
  void _swap(int* a, int* b);
  void _swap(int& a, int& b);
  auto _returnAuto()->int;
};

#endif


```

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: learnauto.cpp
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has learnauto class definition
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "learnauto.h"

void learnauto::basicDataType() {
  {
    bool b = true;
    cout << "size of bool " << sizeof(bool) << " type of b is " << typeid(b).name() << endl;
    char c = 'a';
    cout << "size of char " << sizeof(char) << " type of c is " << typeid(c).name() << endl;
    unsigned int n = 16;
    cout << "size of bool " << sizeof(unsigned int) << " type of n is " << typeid(n).name() << endl;
    int m = 16;
    cout << "size of int " << sizeof(int) << " type of m is " << typeid(m).name() << endl;
    long l = 898901;
    cout << "size of long " << sizeof(long) << " type of l is " << typeid(l).name() << endl;
    long long h = 12345678ll;
    cout << "size of long long " << sizeof(long long) << " type of h is " << typeid(h).name() << endl;
    float f = 7.8f;
    cout << "size of float " << sizeof(float) << " type of f is " << typeid(f).name() << endl;
    double d = 3.141456;
    cout << "size of double " << sizeof(double) << " type of d is " << typeid(d).name() << endl;
  }

  {
    //To show you don't require data type at all
    auto b = true;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    auto c = 'a';
    cout << "type of c is " << typeid(c).name() << " Size = " << sizeof(c) << "
```

```
        Bytes. Value = " << c << endl;
    auto n = 16u; //unsigned int
    cout << "type of n is " << typeid(n).name() << " Size = " << sizeof(n) << " Bytes. Value = " << n << endl;
    auto m = 16;//signed int
    cout << "type of m is " << typeid(m).name() << " Size = " << sizeof(m) << " Bytes. Value = " << m << endl;
    auto l = 898901;//signed long
    cout << "type of l is " << typeid(l).name() << " Size = " << sizeof(l) << " Bytes. Value = " << l << endl;
    auto h = 12345678ll; //signed long long
    cout << "type of h is " << typeid(h).name() << " Size = " << sizeof(h) << " Bytes. Value = " << h << endl;
    auto u = 12345678ull; //unsigned long long
    cout << "type of u is " << typeid(u).name() << " Size = " << sizeof(u) << " Bytes. Value = " << u << endl;
    auto f = 7.8f;
    cout << "type of f is " << typeid(f).name() << " Size = " << sizeof(f) << " Bytes. Value = " << f << endl;
    auto d = 3.141456;
    cout << "type of d is " << typeid(d).name() << " Size = " << sizeof(d) << " Bytes. Value = " << d << endl;
  }

  {
    //To show auto requires initialization not assignment
    double b; //Compiles but d is garbage
    b = 3.14545;
    //auto c;  Will Not compile
    auto c = 3.141456;
  }

  {
    //To show you cannot change data type when once you declared
    //Opposite to Python
    auto b = 3;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    //warning C4244: '=': conversion from 'double' to 'int', possible loss of data
    b = 3.141456;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    b = 5.141456;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
    //Note: There is no context in which this conversion is possible
    //b = "UCSC";
    //cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << 
```

```
        " Bytes. Value = " << b << endl;
  }

  {
    //Pointers to auto
    auto i = 8;
    auto* p = &i;
    cout << " i = " << i << " " << *p << endl;
  }

  {
    //aliases to auto
    auto d = 8.98978;
    auto& p = d;
    cout << " d = " << d << " " << p << endl;
  }

  {
    //b is identified as const char * NOT std::string
    auto b("jag");
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }

  {
    //b is identified  std::string
    //C++ 14 and above
    auto b = "jag"s;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
}

/*-----------------------------------------------------------------
auto as return type
-----------------------------------------------------------------*/
auto learnauto::_returnAuto()->int {
  return 100;
}


/*-----------------------------------------------------------------
auto as return type
-----------------------------------------------------------------*/
void learnauto::testReturnAuto() {
  {
    auto b = _returnAuto();
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
```

```
  }

}

#if 0
/*-----------------------------------------------------------------
test swap
error C3533: a parameter cannot have a type that contains 'auto'
-----------------------------------------------------------------*/
void learnauto::_swap(auto* a, auto* b) {
  auto t = *a;
  *a = *b;
  *b = t;
}
#endif

/*-----------------------------------------------------------------
test swap
-----------------------------------------------------------------*/
void learnauto::_swap(int* a, int* b) {
  auto t = *a;
  *a = *b;
  *b = t;
}

#if 0
/*-----------------------------------------------------------------
test swap
error C3533: a parameter cannot have a type that contains 'auto'
-----------------------------------------------------------------*/
void learnauto::_swap(auto& a, auto& b) {
  auto t = a;
  a = b;
  b = t;
}
#endif

/*-----------------------------------------------------------------
test swap
-----------------------------------------------------------------*/
void learnauto::_swap(int& a, int& b) {
  auto t = a;
  a = b;
  b = t;
}

/*-----------------------------------------------------------------
test swap
-----------------------------------------------------------------*/
```

VASUDEVAMURTHY

```
void learnauto::testSwap() {
  auto a{ 10 };
  auto b{ 20 };
  cout << "Before a = " << a << " b = " << b << endl;
  _swap(&a, &b);
  cout << "After  a = " << a << " b = " << b << endl;
  _swap(a, b);
  cout << "Again  a = " << a << " b = " << b << endl;
}

/*-----------------------------------------------------------------
Initalizing an array
-----------------------------------------------------------------*/
void learnauto::arrayInitialization() {
  {
    double temp[] = { 89.9, 78.9, 56.0 };
    int size = int(sizeof(temp) / sizeof(double));

    for (int i = 0; i < size; ++i) {
      cout << "temp[" << i << "]= " << temp[i] << endl;
    }
  }

  {
    //Why can't I create an array of automatic variables?
    //error C3318: 'auto []': an array cannot have an element type that contains 'auto'
    //auto temp[] = { 89.9, 78.9, 56.0 }; //This is Not possible
  }

  {
    //Only way to do is through initialization list
    auto temp = { 89.9, 78.9, 56.0, 98.0 }; //Vector of doubles
    auto size = temp.size();
    auto i = 0;
    for (auto o : temp) {
      cout << "temp[" << i++ << "]= " << o << endl;
    }
  }

  {
    //error C2676: binary '[': 'std::initializer_list<double>' does not define this
    //operator or a conversion to a type acceptable to the predefined operator
    auto temp = { 89.9, 78.9, 56.0, 98.0 }; //Vector of doubles
    auto size = temp.size();
    for (auto i = 0; i < size; ++i) {
      //This wont compile does not define []
      //cout << "temp[" << i++ << "]= " << temp[i] << endl;
```

```
    }
  }
}

/*-----------------------------------------------------------------
objects
-----------------------------------------------------------------*/
void learnauto::objects() {
  class obj {
    int _x;
  public:
    obj(int x = 4) :_x(x) {}
    void p() const {
      cout << " obj val = " << _x << endl;
    }
  };

  {
    //on stack
    obj p;
    //auto q(100) ;
    auto q(obj(78));
    p.p();
    q.p();
  }

  {
    //on heap
    obj* p = new obj();
    auto q = new obj(100);
    p->p();
    q->p();
    delete p;
    delete q;
  }

  {
    vector <obj> p;
    for (auto i = 0; i < 10; ++i) {
      p.push_back(i);
    }
    {
      auto itt = p.end();
      while (itt != p.begin()) {
        auto& o = *(--itt);
        o.p();
      }
    }
    {
```

```
      for (auto& o : p) {
        o.p();
      }
    }
  }
}

//EOF
```

VASUDEVAMURTHY

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: learnautotest.cpp

On linux:
g++ learnauto.cpp learnautotest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test learnauto object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "learnauto.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  learnauto a;
  a.basicDataType();
  a.testReturnAuto();
  a.testSwap();
  a.arrayInitialization();
  a.objects();
}


/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testbed();
  return 0;
}

//EOF
```

1.8 Initializer list assignment

Initializer list assignments

```
void initializaton::primTypes() {
  {
    //C way
    int b; //declaration
    b = 10 ; //assignment
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {
    //C++ 11 way
    auto b = 10 ; //assignment to an object
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {

    auto b(10) ; //assignment to an object
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {
    //C++ 11
    auto b{10} ; // {10} IS CALLED Initializer list Initialzation FIRST CHOICE
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {
    //C++ 11 prefered way
    double b{10.5} ;
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
}
```

Figure 1.13: Basic type

VASUDEVAMURTHY

Initializer list assignments

```
void initializaton::arrayS() {
  {
    //OLD WAY
    int a[] = { 100, 200, 300, 400 };
    int n = sizeof(a)/sizeof(int);
    for (int i = 0; i < n; ++i) {
      cout << "a[" << i << "] = " << a[i] << " " ;
    }
    cout << endl ;
  }
  {
    int a[] = { 100, 200, 300, 400 };
    int i = 0 ;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " " ;
    }
    cout << endl ;
  }

  {
    //C++ 11 prefered way
    // { 100, 200, 300, 400 } IS CALLED Initalizer list
    // Initalizer list Initialzation FIRST CHOICE
    int a[] { 100, 200, 300, 400 };
    int i = 0 ;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " " ;
    }
    cout << endl ;
  }

  {
    //Vector
    vector<int> a;
    a.push_back(100);
    a.push_back(200);
    a.push_back(300);
    a.push_back(400);
    int i = 0 ;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " " ;
    }
    cout << endl ;
  }

  {
    //C++ 11 old
    vector<int> a = { 100, 200, 300, 400 };
    int i = 0 ;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " " ;
    }
    cout << endl ;
  }

  {
    //C++ 11 prefered way
    // { 100, 200, 300, 400 } IS CALLED Initalizer list
    // Initalizer list Initialzation FIRST CHOICE
    // C++11 Initilaizor list constructor
    //All stl container is modified
    vector<int> a { 100, 200, 300, 400 };
    int i = 0 ;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " " ;
    }
    cout << endl ;
  }
}
```

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

VASUDEVAMURTHY

Initializer list assignments

```
void initializaton::UDT() {
  class myarray {
  public:
    myarray(const initializer_list<int>& v) {
      for (auto t : v) {
        _a.push_back(t);
      }
    }
    auto getSize() const {
      return _a.size() ;
    }
    auto getI(int i) const {
      return _a[i] ;
    }
  private:
    vector<int> _a;
  };

  {
    myarray a = { 100, 200, 300, 400 };
    auto n = a.getSize() ;
    for (auto i = 0; i < n; ++i) {
      cout << "a[" << i << "] = " << a.getI(i) << " " ;
    }
    cout << endl ;
  }


  //C++ 11 prefered way
  // { 100, 200, 300, 400 } IS CALLED Initalizer list
  // Initalizer list Initialzation FIRST CHOICE
  {
    myarray a { 100, 200, 300, 400 };
    auto n = a.getSize() ;
    for (auto i = 0; i < n; ++i) {
      cout << "a[" << i << "] = " << a.getI(i) << " " ;
    }
    cout << endl ;
  }
}
```

```
/*-----------------------------------------------------------
aggregateInitialization  VS ConstructorInitialization

Uniform Initilaization search order
1. Initalizer list Initialzation FIRST CHOICE
2. Constructor initialization SECOND CHOICE
3. Aggregate initialization, Old C++. THIRD CHOICE
-----------------------------------------------------------
void initializaton::aggregateInitializationConstructorInitialization() {
  class dog{
  public:
    int age;
    string name;
  };

  class cat{
  public:
    cat(int a, const string& s) :_age(a), _name(s) {
      cout << "SECOND PREFERENCE In cat(int a, const string& s) c
    };

    cat(const initializer_list<int>& s) {
      cout << "FIRST PREFERENCE initializer_list<int>&\n";
    }

  private:
    int _age;
    string _name;
  };

  {
    //THIRD PREFERENCE Aggregate initialization
    //Note public in class
    dog d = { 5, "tommy" };
    dog r {5, "tommy" };
  }

  {
    //3 Constructor initialization SECOND CHOICE
    //Note private in class
    cat c { 6, "meow" };
    cat s { 1 };
  }
}
```

Figure 1.15: User defined types

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: initializaton.h
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has initializaton class declaration
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#ifndef initializaton_H
#define initializaton_H

#include "../util/util.h"

/*----------------------------------------------------------------
Declaration of initializaton class
----------------------------------------------------------------*/
class initializaton {
public:
  void primTypes();
  void arrayS();
  void UDT();
  void aggregateInitializationConstructorInitialization();
private:

};

#endif

```

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: initializaton.cpp
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has initializaton class definition
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "initializaton.h"

/*----------------------------------------------------------------
prim type
----------------------------------------------------------------*/
void initializaton::primTypes() {
  {
    //C way
    int b; //declaration
    b = 10; //assignment
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {
    //C++ 11 way
    auto b = 10; //assignment to an object
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {

    auto b(10); //assignment to an object
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {
    //C++ 11
    auto b{ 10 }; // {10} IS CALLED Initializer list Initialzation FIRST CHOICE
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
  {
    //C++ 11 prefered way
    double b{ 10.5 };
    cout << "type of b is " << typeid(b).name() << " Size = " << sizeof(b) << " Bytes. Value = " << b << endl;
  }
```

```
}

/*-----------------------------------------------------------------
arrays
-----------------------------------------------------------------*/
void initializaton::arrayS() {
  {
    //OLD WAY
    int a[] = { 100, 200, 300, 400 };
    int n = sizeof(a) / sizeof(int);
    for (int i = 0; i < n; ++i) {
      cout << "a[" << i << "] = " << a[i] << " ";
    }
    cout << endl;
  }
  {
    int a[] = { 100, 200, 300, 400 };
    int i = 0;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " ";
    }
    cout << endl;
  }

  {
    //C++ 11 prefered way
    // { 100, 200, 300, 400 } IS CALLED Initalizer list
    // Initalizer list Initialzation FIRST CHOICE
    int a[]{ 100, 200, 300, 400 };
    int i = 0;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " ";
    }
    cout << endl;
  }

  {
    //Vector
    vector<int> a;
    a.push_back(100);
    a.push_back(200);
    a.push_back(300);
    a.push_back(400);
    int i = 0;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " ";
    }
    cout << endl;
  }
```

```

  {
    //C++ 11 old
    vector<int> a = { 100, 200, 300, 400 };
    int i = 0;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " ";
    }
    cout << endl;
  }

  {
    //C++ 11 prefered way
    // { 100, 200, 300, 400 } IS CALLED Initalizer list
    // Initalizer list Initialzation FIRST CHOICE
    // C++11 Initilaizor list constructor
    //All stl container is modified
    vector<int> a{ 100, 200, 300, 400 };
    int i = 0;
    for (auto e : a) {
      cout << "a[" << i++ << "] = " << e << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
UDT
-----------------------------------------------------------------*/
void initializaton::UDT() {
  class myarray {
  public:
    myarray(const initializer_list<int>& v) {
      for (auto t : v) {
        _a.push_back(t);
      }
    }
    auto getSize() const {
      return _a.size();
    }
    auto getI(int i) const {
      return _a[i];
    }
  private:
    vector<int> _a;
  };

  {
    myarray a = { 100, 200, 300, 400 };
```

```
    auto n = a.getSize();
    for (auto i = 0; i < n; ++i) {
      cout << "a[" << i << "] = " << a.getI(i) << " ";
    }
    cout << endl;
  }


  //C++ 11 prefered way
  // { 100, 200, 300, 400 } IS CALLED Initalizer list
  // Initalizer list Initialzation FIRST CHOICE
  {
    myarray a{ 100, 200, 300, 400 };
    auto n = a.getSize();
    for (auto i = 0; i < n; ++i) {
      cout << "a[" << i << "] = " << a.getI(i) << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
aggregateInitialization  VS ConstructorInitialization

Uniform Initilaization search order
1. Initalizer list Initialzation FIRST CHOICE
2. Constructor initialization SECOND CHOICE
3. Aggregate initialization, Old C++. THIRD CHOICE
-----------------------------------------------------------------*/
void initializaton::aggregateInitializationConstructorInitialization() {
  class dog {
  public:
    int age {10} ;
    int h = 20 ;
    //int w(0) ; //This wont compile
    string name {"none"} ;
    void P(const string& t) const {
      cout << t << " age = " << age << " b = " << h << " name = " << name <<
       endl ;
    }
  };

  class cat {
  public:
    cat(int a, const string& s) :_age(a), _name(s) {
      cout << "SECOND PREFERENCE In cat(int a, const string& s) constructor"  <<
        endl;
    };

```

```
    cat(const initializer_list<int>& s) {
      cout << "FIRST PREFERENCE initializer_list<int>&\n";
    }

  private:
    int _age;
    string _name;
  };

  {
    //THIRD PREFERENCE Aggregate initialization
    //Note public in class
    dog m;
    m.P("dog m") ;
    dog d = { 5,3, "tommy" };
    d.P("dog d") ;
    dog r{ 5,18, "dommy" };
    r.P("dog r") ;
  }

  {
    //3 Constructor initialization SECOND CHOICE
    //Note private in class
    cat c{ 6, "meow" };
    cat s{ 1 };
  }
}


//https://www.geeksforgeeks.org/when-do-we-use-initializer-list-in-c/
//EOF
```

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: initializatontest.cpp

On linux:
g++ initializaton.cpp initializatontest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test initializaton object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "initializaton.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  initializaton a;
  a.primTypes();
  a.arrayS();
  a.UDT();
  a.aggregateInitializationConstructorInitialization();
}


/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testbed();
  return 0;
}

//EOF
```

1.9 Arithmetic operations

Figure 1.16: Arithmetic unary operators

VASUDEVAMURTHY

1.9.1 C++code for illustrating arithmetic operations

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: arithmetic.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has arithmetic class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef ARITHMETIC_H
#define ARITHMETIC_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of arithmetic class
-----------------------------------------------------------------*/
class arithmetic{
public:
  void test_unary();
  void test_increment();
  void test_decrement();
  void test_int();
  void test_float();
  void test_double();

private:

};

#endif
//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: arithmetic.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has arithmetic class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "arithmetic.h"

/*-----------------------------------------------------------------
Definition of routines of arithmetic class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
a = -10
t = 10
Note that a is not changed after the operation -a
After -a, a = -10
-t =  -10 and t = 10
-----------------------------------------------------------------*/
void arithmetic::test_unary() {
  int a = -10 ;
  cout << "a = " << a << endl ;
  int t = -a ;
  cout << "t = " << t << endl ;
  cout << "Note that a is not changed after the operation -a\n" ;
  cout << "After -a, a = " << a << endl ;
  cout << "-t =  " << -t << " and t = " << t << endl ;
}

/*-----------------------------------------------------------------
1. c =  5
2. c =  6
3. c =  6
4. c =  7
5. c =  8
6. c =  8
-----------------------------------------------------------------*/
void arithmetic::test_increment() {
  int c = 5 ;
  cout << " 1. c =  " << c << endl ;
  c++ ;
  cout << " 2. c =  " << c << endl ;
  cout << " 3. c =  " << c++ << endl ;
  cout << " 4. c =  " << c << endl ;
  cout << " 5. c =  " << ++c << endl ;
  cout << " 6. c =  " << c << endl ;
}

/*-----------------------------------------------------------------
1. c =  5
2. c =  4
3. c =  4
4. c =  3
5. c =  2
6. c =  2
-----------------------------------------------------------------*/
void arithmetic::test_decrement() {
  int c = 5 ;
  cout << " 1. c =  " << c << endl ;
  c-- ;
```

```
  cout << " 2. c =  " << c << endl ;
  cout << " 3. c =  " << c-- << endl ;
  cout << " 4. c =  " << c << endl ;
  cout << " 5. c =  " << --c << endl ;
  cout << " 6. c =  " << c << endl ;
}

/*-----------------------------------------------------------------
a = 10 b = 3
1. a+b =  13
2. a-b =  7
3. a*b =  30
4. a/b =  3
5. a%b =  1
a = 10 b = 3
7. c =  13
8. c =  39
9. c =  36
10. c =  108
-----------------------------------------------------------------*/
void arithmetic::test_int() {
  int a = 10 ;
  int b = 3 ;
  cout << "a = " << a << " b = " << b << endl ;
  cout << " 1. a+b =  " << a+b << endl ;
  cout << " 2. a-b =  " << a-b << endl ;
  cout << " 3. a*b =  " << a*b << endl ;
  cout << " 4. a/b =  " << a/b << endl ;
  cout << " 5. a%b =  " << a%b << endl ;
  cout << "a = " << a << " b = " << b << endl ;
  int c = 10 ;
  c += 3 ;
  cout << " 7. c =  " << c << endl ;
  c *= 3 ;
  cout << " 8. c =  " << c << endl ;
  c -= 3 ;
  cout << " 9. c =  " << c << endl ;
  c *= 3 ;
  cout << " 10. c =  " << c << endl ;
}

/*-----------------------------------------------------------------
a = 10.3 b = 3.21
1. a+b =  13.51
2. a-b =  7.09
3. a*b =  33.063
4. a/b =  3.20872
-----------------------------------------------------------------*/
void arithmetic::test_float() {
  float a = 10.3f ;
  float b = 3.21f ;
  cout << "a = " << a << " b = " << b << endl ;
  cout << " 1. a+b =  " << a+b << endl ;
  cout << " 2. a-b =  " << a-b << endl ;
  cout << " 3. a*b =  " << a*b << endl ;
  cout << " 4. a/b =  " << a/b << endl ;
  //cout << " 5. a%b =  " << a%b << endl ;
  // error C2296: '%' : illegal, left operand has type 'float'
}

/*-----------------------------------------------------------------
a = 10.3 b = 3.21
1. a+b =  13.51
2. a-b =  7.09
3. a*b =  33.063
4. a/b =  3.20872
```


VAMUDEVAMURTHY

```
---------------------------------------------------------------*/
void arithmetic::test_double() {
  double a = 10.3 ;
  double b = 3.21 ;
  cout << "a = " << a << " b = " << b << endl ;
  cout << " 1. a+b =  " << a+b << endl ;
  cout << " 2. a-b =  " << a-b << endl ;
  cout << " 3. a*b =  " << a*b << endl ;
  cout << " 4. a/b =  " << a/b << endl ;
  //cout << " 5. a%b =  " << a%b << endl ;
  // error C2296: '%' : illegal, left operand has type 'float'
}


//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: arithmetictest.cpp

On linux:
g++ arithmetic.cpp arithmetictest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test arithmetic object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "arithmetic.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  arithmetic a ;
  a.test_unary() ;
  a.test_increment();
  a.test_decrement();
  a.test_int();
  a.test_float();
  a.test_double();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

Figure 1.17: Arithmetic binary operators

1.10 Relational operations

VASUDEVAMURTHY

1.10.1 C++code for illustrating relational operations

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: relational.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has relational class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef RELATIONAL_H
#define RELATIONAL_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of relational class
-----------------------------------------------------------------*/
class relational{
public:
  void test_relational();
  void test_ternery();

private:

};

#endif
//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: relational.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has relational class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "relational.h"

/*-----------------------------------------------------------------
Definition of routines of relational class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
a = 7 b = 12
1. a < b = true
2. a <= b = true
3. a > b = false
4. a >= b = false
5. a == b = false
6. a != b = true
-----------------------------------------------------------------*/
void relational::test_relational() {
  int  a  = 7;
  int b = 12 ;
  cout << boolalpha ; //displays boolean as true or false
  cout << "a = " << a << " b = "<< b << endl ;
  cout << "1. a < b = " << (a < b) << endl ;
  cout << "2. a <= b = "<< (a <= b) << endl ;
  cout << "3. a > b = "<< (a > b) << endl ;
  cout << "4. a >= b = "<< (a >= b) << endl ;
  cout << "5. a == b = "<< (a == b) << endl ;
  cout << "6. a != b = "<< (a != b) << endl ;
  cout.unsetf(ios::boolalpha) ; //unset boolalpha
}

/*-----------------------------------------------------------------
a = 10 b = 20
1. c = 77
a = 20 b = 10
2. c = 5
-----------------------------------------------------------------*/
void relational::test_ternery() {
  int a = 10 ;
  int b = 20 ;
  cout << "a = " << a << " b = "<< b << endl ;
  int c = (a < b) ? (a + 67) : (b -5) ;
  cout << "1. c = " << c << endl ;
  a = 20 ;
  b = 10 ;
  cout << "a = " << a << " b = "<< b << endl ;
  c = (a < b) ? (a + 67) : (b -5) ;
  cout << "2. c = " << c << endl ;
}


//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: relationaltest.cpp

On linux:
g++ relational.cpp relationaltest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test relational object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "relational.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  relational a ;
  a.test_relational();
  a.test_ternery();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

1.11 Logical operations

Figure 1.19: Logical operations

1.11.1 C++code for illustrating logical operations

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: logical.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has logical class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef logical_H
#define logical_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of logical class
-----------------------------------------------------------------*/
class logical{
public:
  void test_logical();
  void test_comma();

private:

};

#endif
//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: logical.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has logical class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "logical.h"

/*-----------------------------------------------------------------
Definition of routines of logical class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
a = 7 b = 12 c = 5 d = 0
1. (a < b) && (b > c) = true
2. (a < b) && (b < c) = false
3. (a < b) || (b < c) = true
4. (a > b) || (b < c) = false
5. (!a) = false
6. (d) = 0
7. (d) = false
8. (!d) = true
Note that d is not changed after  !d. d = 0
-----------------------------------------------------------------*/
void logical::test_logical() {
  int  a  = 7;
  int b = 12 ;
  int c = 5 ;
  int d = 0 ;
  cout << boolalpha ; //displays boolean as true or false
  cout << "a = " << a << " b = "<< b << " c = " << c << " d = " << d << endl ;
  cout << "1. (a < b) && (b > c) = " <<  ((a < b) && (b > c)) << endl ;
  cout << "2. (a < b) && (b < c) = " <<  ((a < b) && (b < c)) << endl ;
  cout << "3. (a < b) || (b < c) = " <<  ((a < b) || (b < c)) << endl ;
  cout << "4. (a > b) || (b < c) = " <<  ((a > b) || (b < c)) << endl ;
  cout << "5. (!a) = " << (!a) << endl ;
  cout << "6. (d) = " << (d) << endl ;
  cout << "7. (d) = " << bool(d) << endl ;
  cout << "8. (!d) = " << (!d) << endl ;
  cout << "Note that d is not changed after  !d. d = " << d << endl ;
  cout.unsetf(ios::boolalpha) ; //unset boolalpha
}

/*-----------------------------------------------------------------
a = 7 b = 12
1. k=(a,b) = 12 //Evalauted from left to right;final 12 is assigned to k
2. k=(b,a) = 7
-----------------------------------------------------------------*/
void logical::test_comma() {
  int  a  = 7;
  int b = 12 ;
  cout << "a = " << a << " b = "<< b << endl ;
  int k = (a,b) ;
  cout << "1. k=(a,b) = " <<  k << endl ;
  k = (b,a) ;
  cout << "2. k=(b,a) = " <<  k << endl ;
}

//EOF
```

VASU - VAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: logicaltest.cpp

On linux:
g++ logical.cpp logicaltest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test logical object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "logical.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  logical a ;
  a.test_logical();
  a.test_comma();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF


```

VASUDEVAMURTHY

Relational operations

```
< <= > >= == !=
```

Ternary operation

```
if (s)
   f = a ; f = (s) ? a : b
else
   f = b ;
```

Figure 1.18: Relational operations

1.12 Bitwise operations

VASUDEVAMURTHY

Bitwise Unary operations

~j Unary bitwise complement

j will not be changed. The answer on the RHS will be changed

```
int a = 0 ;   //a is 0
int t = ~a ;  // t is ffffffff
  //a is 0 at this point
```

Bitwise Binary operations

```
a & b   Bitwise AND
a | b   Bitwise OR
a ^ b   Bitwise XOR
```

Bitwise Shift operators

```
a << k   Left shift a, k times. Fill lost space by 0
a >> k   Right shift a, k times. Lost space on MSB
                                 is 0 or 1(arithmetic shift)
```

```
int a = 1 ; //a is 1
int t = a << 3 ; //t is 8
//a is 1 at this point
```

```
int a = -3 ;
3 is 011. 2'complement is: 100
                             1
                           ----
                           101
1111 1111 1111 1111 1111 1111 1111 1101
Right shift by 1
1111 1111 1111 1111 1111 1111 1111 1110
```

Figure 1.20: Bitwise operations

VASUDEVAMURTHY

1.12.1 C++code for illustrating bitwise operations

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: bitwise.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has bitwise class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef bitwise_H
#define bitwise_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of bitwise class
-----------------------------------------------------------------*/
class bitwise{
public:
  void test_bitwise();
  void test_shift();

private:

};

#endif
//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: bitwise.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has bitwise class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "bitwise.h"

/*-----------------------------------------------------------------
Definition of routines of bitwise class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All answers are in hexadecimal
1. a = 0
2. ~a = ffffffff
Note that a NOT is changed,. Only the RHS is changed
2a. After ~a, a = 0
3. a = 1
4. b = 2
5. (a & b) = 0
6. (a ! b) = 3
7. (a ^ b) = 3
8. (a & c) = 1
9. (a ! c) = 3
10. (a ^ c) = 2
-----------------------------------------------------------------*/
void bitwise::test_bitwise() {
  int a = 0 ;
  cout << hex ;
  cout << "All answers are in hexadecimal" << endl ;
  cout << "1. a = " << a << endl ;
  cout << "2. ~a = " << ~a << endl ;
  cout << "Note that a NOT is changed,. Only the RHS is changed\n" ;
  cout << "2a. After ~a, a = " << a << endl ;
  a = 1 ;
  int b = 2 ;
  cout << "3. a = " << a << endl ;
  cout << "4. b = " << b << endl ;
  cout << "5. (a & b) = " << (a & b) << endl ;
  cout << "6. (a ! b) = " << (a | b) << endl ;
  cout << "7. (a ^ b) = " << (a ^ b) << endl ;
  int c = 3 ;
  cout << "8. (a & c) = " << (a & c) << endl ;
  cout << "9. (a ! c) = " << (a | c) << endl ;
  cout << "10. (a ^ c) = " << (a ^ c) << endl ;
  cout.unsetf(ios::hex);
}

/*-----------------------------------------------------------------
BEGIN a = 1
All answers are in hexadecimal
Note that a << 3 will NOT effect a. The result will be shifted by 3
a = 1
a << 3 = 8
after shifting a = 1
shift a = 1 left by one = 2
shift a = 2 left by one = 4
shift a = 4 left by one = 8
shift a = 8 left by one = 10
```


VASUDEVAMURTHY

```
shift a = 10 left by one = 20
-----------------------
shift a = 20 right by one = 10
shift a = 10 right by one = 8
shift a = 8 right by one = 4
shift a = 4 right by one = 2
shift a = 2 right by one = 1
shift a = fffffffd right by one = fffffffe
-----------------------------------------------------------------*/
void bitwise::test_shift() {
  int a = 1 ;
  cout << "BEGIN a = " << a << endl ;

  cout << hex ;
  cout << "All answers are in hexadecimal" << endl ;
  cout << "Note that a << 3 will NOT effect a. The result will be shifted by 3\n" ;
  cout << "a = " << a << endl ;
  cout << "a << 3 = " << (a << 3) << endl ;
  cout << "after shifting a = " << a << endl ;
  for (int i = 0 ; i < 5 ; i++) {
    cout << "shift a = " << a << " left by one = " ;
    cout << (a = a << 1) << endl ;
  }
  cout << "-----------------------" << endl ;
  for (int i = 0 ; i < 5 ; i++) {
    cout << "shift a = " << a << " right by one = " ;
    cout << (a = a >> 1) << endl ;
  }
  a = -3 ;
  cout << "shift a = " << a << " right by one = " ;
  cout << (a = a >> 1) << endl ;
  cout.unsetf(ios::hex);
}
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: bitwisetest.cpp

On linux:
g++ bitwise.cpp bitwisetest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test bitwise object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "bitwise.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  bitwise a ;
  a.test_bitwise();
  a.test_shift();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

1.13 Assignment operations

Assignment

=	a = b

Additional assignment

+=	a+=b	a = a + b
-=	a-=b	a = a - b
*=	a*=b	a = a * b
/=	a/=b	a = a / b
%=	a%=b	a = a % b

Bitwise assignment

&=	a&=b	a = a & b
!=	a!=b	a = a | b
^=	a^=b	a = a ^ b

Figure 1.21: Assignment operations

1.14 Control statements

VASUDEVAMURTHY

Figure 1.22: Control statements

VASUDEVAMURTHY

Figure 1.23: switch statement

VASUDEVAMURTHY

1.14.1 C++code for illustrating control statements

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: control.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has control class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef control_H
#define control_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of control class
-----------------------------------------------------------------*/
class control{
public:
  void test_block();
  void test_if();
  void test_if_else() ;
  void test_for() ;
  void test_while() ;
  void test_do_while() ;
  void test_need_for_switch() ;
  void test_switch() ;
  void test_switch1() ;
private:

};

#endif
//EOF
```

VASUDEVAMURTHY

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: control.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has control class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "control.h"

/*-----------------------------------------------------------------
Definition of routines of control class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
i = 88
i = 77
i = 88
-----------------------------------------------------------------*/
void control::test_block() {
  int i = 88 ;
  cout <<  " i = " << i << endl;
  {
    int i = 77 ;
    cout <<  " i = " << i << endl;
    int j = 20 ;
  }
  cout <<  " i = " << i << endl;
  //cout <<  " j = " << j << endl; //error C2065: 'j' : undeclared identifier
}

/*-----------------------------------------------------------------
i = 81
80 < 85
i = 81
-----------------------------------------------------------------*/
void control::test_if() {
  int i = 80 ;
  if (i > 85)
    cout <<  i << " > "  << "85\n" ;
  i++ ;
  cout << "i = " << i <<  endl ;
  i-- ;
  if (i < 85) {
    cout <<  i << " < "  << "85\n" ;
```

```
    i++ ;
  }
  cout << "i = " << i <<  endl ;
}

/*-----------------------------------------------------------------
max = 50
-----------------------------------------------------------------*/
void control::test_if_else() {
  int i = 30;
  int j = 10 ;
  int k = 50 ;
  int max = -1 ;
  if (i > j) {
    // i is greater
    if (i > k) {
      // i is greater
      max = i ;
    }else {
      //k is greater
      max = k ;
    }
  }else {
    //j is greater
    if (j > k) {
      // j is greater
      max = j;
    }else {
      //k is greater
      max = k ;
    }
  }
  cout << "max = " << max << endl ;
}

/*-----------------------------------------------------------------
100 101 102 103 104 105 106 107 108 109
100 103 106 109

100 99 98 97 96 95 94 93 92 91
50 52 54 56
50 52 54 56
50 52 54 56
-----------------------------------------------------------------*/
void control::test_for() {
  for (int i = 100; i < 110; i++) {
    cout << i << " " ;
  }
  cout << endl ;
```

```
  for (int i = 100; i < 110; i=i+3){
    cout << i << " " ;
  }
  cout << endl ;

  //to demonstrate i is dead here
  //cout << i << endl ; //error C2065: 'i' : undeclared identifier

#if 0
  //What happens ?
  for (int i = 100; i < 110; i--){
    cout << i << " " ;
  }
  cout << endl ;
#endif

  for (int i = 100; i < 90; i--) {
    cout << i << " " ;
  }
  cout << endl ;

  for (int i = 100; i > 90; i--) {
    cout << i << " " ;
  }
  cout << endl ;
  {
    int i = 50 ;
    for (; i < 57; i=i+2) {
      cout << i << " " ;
    }
    cout << endl ;
  }
  {
    int i = 50 ;
    for (; i < 57;) {
      cout << i << " " ;
      i = i + 2 ;
    }
    cout << endl ;
  }
  {
    for (int i = 0; i < 10; i++) {
      cout << " i = " << i << endl ;
      if (i == 5) {
        break ;
      }
    }

  }
```

```
148   {
149     int i = 50 ;
150     for (;;) {
151       cout << i << " " ;
152       i = i + 2 ;
153       if (i >= 57) {
154         break ;
155       }
156     }
157     cout << endl ;
158   }
159 }
160
161 /*-----------------------------------------------------------------
162 100 101 102 103 104 105 106 107 108 109
163 100 103 106 109
164
165 100 99 98 97 96 95 94 93 92 91
166 50 52 54 56
167 50 52 54 56
168 50 52 54 56
169 -----------------------------------------------------------------*/
170 void control::test_while() {
171   {
172     int i = 100 ;
173     while (i < 110) {
174       cout << i << " " ;
175       i++ ;
176     }
177     cout << endl ;
178   }
179   {
180     int i = 100 ;
181     while (i < 110) {
182       cout << i << " " ;
183       i = i + 3  ;
184     }
185     cout << endl ;
186   }
187
188 #if 0
189   //What happens ?
190   int i = 100 ;
191   while (i < 110) {
192     cout << i << " " ;
193     i-- ;
194   }
195   cout << endl ;
196 #endif
```

```
  {
    int i = 100 ;
    while (i < 90) {
      cout << i << " " ;
      i-- ;
    }
    cout << endl ;
  }
  {
    int i = 100 ;
    while ( i > 90) {
      cout << i << " " ;
      i-- ;
    }
    cout << endl ;
  }
  {
    int i = 50 ;
    while (i < 57) {
      cout << i << " " ;
      i = i + 2;
    }
    cout << endl ;
  }
  {
    int i = 50 ;
    while (i < 57) {
      cout << i << " " ;
      i = i + 2 ;
    }
    cout << endl ;
  }
  {
    int i = 50 ;
    while(true) {
      cout << i << " " ;
      i = i + 2 ;
      if (i >= 57) {
        break ;
      }
    }
    cout << endl ;
  }
}

/*-----------------------------------------------------------------
100 101 102 103 104 105 106 107 108 109
100 103 106 109
100
```

```
100 99 98 97 96 95 94 93 92 91
50 52 54 56
50 52 54 56
50 52 54 56
-----------------------------------------------------------------*/
void control::test_do_while() {
  {
    int i = 100 ;
    do {
      cout << i << " " ;
      i++ ;
    } while (i < 110) ;
    cout << endl ;
  }
  {
    int i = 100 ;
    do {
      cout << i << " " ;
      i = i + 3  ;
    } while (i < 110);
    cout << endl ;
  }

#if 0
  //What happens ?
  int i = 100 ;
  do  {
    cout << i << " " ;
    i-- ;
  }while (i < 110);
  cout << endl ;
#endif
  {
    int i = 100 ;
    do{
      cout << i << " " ;
      i-- ;
    } while (i < 90) ;
    cout << endl ;
  }
  {
    int i = 100 ;
    do{
      cout << i << " " ;
      i-- ;
    } while ( i > 90) ;
    cout << endl ;
  }
  {
```

```
    int i = 50 ;
    do {
      cout << i << " " ;
      i = i + 2;
    } while (i < 57);
    cout << endl ;
  }

  {
    int i = 50 ;
    do {
      cout << i << " " ;
      i = i + 2 ;
      if (i >= 57) {
        break ;
      }
    } while(true) ;
    cout << endl ;
  }
}

/*-----------------------------------------------------------------
Marks = 7 Grade =  f bonus = 0
-----------------------------------------------------------------*/
void control::test_need_for_switch() {
  int marks = 7 ;
  int bonus  = -1;
  char grade  = ' ' ;
  if (marks > 90)  {
    grade = 'a' ;
    bonus = 5 ;
  } else if (marks > 80){
    grade = 'b' ;
    bonus = 4 ;
  } else if (marks > 70) {
    grade = 'c' ;
    bonus = 3 ;
  } else if (marks > 60){
    grade = 'd' ;
    bonus = 2 ;
  } else if (marks > 50) {
    grade = 'e' ;
    bonus = 1 ;
  } else {
    grade = 'f' ;
    bonus = 0 ;
  }
  cout << "Marks = " << marks << " Grade =  " << grade  << " bonus = " << bonus
    << endl ;
```

```
}

/*-----------------------------------------------------------------
Marks = 7 Grade =  f bonus = 0
-----------------------------------------------------------------*/
void control::test_switch() {
  int marks = 7 ;
  int bonus  = -1;
  char grade  = ' ' ;
  int k = (marks -1) /10 ; //Must get evaluated to an integer value
  switch (k) {
  case 9:
    grade = 'a' ;
    bonus = 5 ;
    break ;
  case 8:
    grade = 'b' ;
    bonus = 4 ;
    break ;
  case 7:
    grade = 'c' ;
    bonus = 3 ;
    break ;
  case 6:
    grade = 'd' ;
    bonus = 2 ;
    break ;
  case 5:
    grade = 'e' ;
    bonus = 1 ;
    break ;
  default:
    grade = 'f' ;
    bonus = 0 ;
    break ;
  }
  cout << "Marks = " << marks << " Grade =  " << grade  << " bonus = " << bonus
    << endl ;
}

/*-----------------------------------------------------------------
Marks = 7 Grade =  d bonus = 2
-----------------------------------------------------------------*/
void control::test_switch1() {
  int marks = 7 ;
  int bonus  = -1;
  char grade  = ' ' ;
  int k = 8 ;
  switch (k) {
```

```
    case 9:
      grade = 'a' ;
      bonus = 5 ;
      break ;
    case 8:
      grade = 'b' ;
      bonus = 4 ;
    case 7:
      grade = 'c' ;
      bonus = 3 ;
    case 6:
      grade = 'd' ;
      bonus = 2 ;
      break ;
    case 5:
      grade = 'e' ;
      bonus = 1 ;
      break ;
    default:
      grade = 'f' ;
      bonus = 0 ;
      break ;
    }
    cout << "Marks = " << marks << " Grade =  " << grade  << " bonus = " << bonus
      << endl ;
  }
```

VASUDEVA

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: controltest.cpp

On linux:
g++ control.cpp controltest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test control object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "control.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  control a ;
  a.test_block() ;
  a.test_if() ;
  a.test_if_else() ;
  a.test_for() ;
  a.test_while() ;
  a.test_do_while() ;
  a.test_need_for_switch() ;
  a.test_switch() ;
  a.test_switch1() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

1.15 Enumerations(enum)

VASUDEVAMURTHY

```
static void using_magic_numbers(int dir) {
  switch (dir) {
  case 1: cout << "Go north\n" ;
          break ;
  case 2: cout << "Go south\n" ;
          break ;
  case 3: cout << "Go west\n" ;
          break ;
  case 4: cout << "Go east\n" ;
          break ;
  }
}
```

Need to document dir
Use of magic numbers

```
#define NORTH 1
#define SOUTH 2
#define EAST 4
#define WEST 3

static void using_define(int dir) {
  switch (dir) {
  case NORTH: cout << "Go north\n" ;
          break ;
  case SOUTH: cout << "Go south\n" ;
          break ;
  case WEST: cout << "Go west\n" ;
          break ;
  case EAST: cout << "Go east\n" ;
          break ;
  }
}
```

No type checking
Handled by c++ reprocessor

Figure 1.24: Need for enumeration

VASUDEVAMURTHY

```
const int NORTH = 1;
const int SOUTH = 2 ;
const int EAST = 4 ;
const int WEST = 3 ;

static void using_consts(int dir) {
  switch (dir) {
  case NORTH: cout << "Go north\n" ;
        break ;
  case SOUTH: cout << "Go south\n" ;
        break ;
  case WEST: cout << "Go west\n" ;
        break ;
  case EAST: cout << "Go east\n" ;
        break ;
  }
}
```

Data type: Int
Information not together

```
enum direction {NORTH=1, SOUTH, EAST, WEST} ;

static void using_enum(direction dir) {
  switch (dir) {
  case NORTH: cout << "Go north\n" ;
        break ;
  case SOUTH: cout << "Go south\n" ;
        break ;
  case WEST: cout << "Go west\n" ;
        break ;
  case EAST: cout << "Go east\n" ;
        break ;
  }
}
```

Create new data type
All info in one place

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Figure 1.25: Enumeration:your own data tpe

VASUDEVAMURTHY

1.15.1 C++code for illustrating enumeration

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: enumeration.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef enumeration_H
#define enumeration_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of enumeration class
-----------------------------------------------------------------*/
class enumeration{
public:
  enum direction{NORTH=1, SOUTH, EAST, WEST} ;
  enum  day{
    mon = 3, tue = 8, wed = 2, thu, fri, sat, sun
  };
  enum  day_of_months{
    jan = 31, feb = 28, mar = 31, apr = 30, may = 31, june = 30,
    july = 31, aug = 31, sep = 30, oct = 31, nov = 30, dec = 31
  } ;
  enum Days {             // Declare enum type Days
    saturday,            // saturday = 0 by default
    sunday = 0,          // sunday = 0 as well
    monday,              // monday = 1
    tuesday,             // tuesday = 2
    wednesday,           // etc.
    thursday,
    friday
  };

  void go_in_this_direction(enumeration::direction dir) ;
  void day_of_the_week(enumeration::day d) ;
  void which_month(enumeration::day_of_months m) ;
  void test_default_enum_starts_with_zero() ;
private:

};

#endif
//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: enumeration.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "enumeration.h"

/*-----------------------------------------------------------------
Definition of routines of enumeration class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
if dir is passed as WEST

Go west
-----------------------------------------------------------------*/
void enumeration::go_in_this_direction(enumeration::direction dir) {
  switch (dir) {
  case NORTH: cout << "Go north\n" ;
    break ;
  case SOUTH: cout << "Go south\n" ;
    break ;
  case WEST: cout << "Go west\n" ;
    break ;
  case EAST: cout << "Go east\n" ;
    break ;
  }
}

/*-----------------------------------------------------------------
if d is passed as sat

Enjoy weekend
go to school
8
2
wed 2 thu = 3 fri 4 sat 5 sun 6
-----------------------------------------------------------------*/
void enumeration::day_of_the_week(enumeration::day d) {
  if (d == sat || d == sun) {
    cout << "Enjoy weekend\n" ;
  }
  int k = sat - 1 ;
  if (k == fri) {
    cout << "go to school\n" ;
  }
  cout << tue << endl ;
  int x = tue ;
  cout << wed << endl ;
  cout << "wed " << wed << " thu = " << thu << " fri " << fri << " sat " << sat << " sun " << sun << endl ;
}

/*-----------------------------------------------------------------
if 'd' is passed as feb

I am in feb
I am in may
-----------------------------------------------------------------*/
```


VASUDEVAMURTHY

```
void enumeration::which_month(enumeration::day_of_months d) {
  //d++ ; error C2676: binary '++' : 'test_month::day_of_months' does not define this operator or a
    conversion to a type acceptable to the predefined operator
  if (d == feb) {
    cout << "I am in feb\n" ;
  }
  enumeration::day_of_months m = may ;
  if (m == may) {
    cout << "I am in may\n" ;
  }
}

/*-----------------------------------------------------------------
Not Monday
-----------------------------------------------------------------*/
void enumeration::test_default_enum_starts_with_zero() {
  enumeration::Days today = sunday;
  switch (today) {
  case 1:
    cout << "It's Monday" << endl;
    break;
  default:
    cout << "Not Monday" << endl;
  }
}

//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: enumerationtest.cpp

On linux:
g++ enumeration.cpp enumerationtest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test enumeration object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "enumeration.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  enumeration a ;
  a.go_in_this_direction(enumeration::direction::WEST);
  a.day_of_the_week(enumeration::day::sat) ;
  a.which_month(enumeration::day_of_months::feb) ;
  a.test_default_enum_starts_with_zero() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF
```

1.16 Introductions to functions

Figure 1.26: Call by value

1.16.1 C++code for illustrating call by value function

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: callbyvalue.h
-------------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has callbyvalue class declaration
-------------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-------------------------------------------------------------------*/
#ifndef callbyvalue_H
#define callbyvalue_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of callbyvalue class
-------------------------------------------------------------------*/
class callbyvalue{
public:
  void no_parameters_function();
  void test_f1() ;
  void call_by_value_function(int coffee, int tea);
  void test_f2();
  int call_by_value_and_return_function(int coffee, int tea);
  void test_f3();
private:

};

#endif
//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: callbyvalue.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "callbyvalue.h"

/*-----------------------------------------------------------------
Definition of routines of callbyvalue class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
global variable to this file. BAD way
-----------------------------------------------------------------*/
static int global_i = 10 ;

/*-----------------------------------------------------------------
This function has no input and output
-----------------------------------------------------------------*/
void callbyvalue::no_parameters_function() {
  cout << "global_i  = " << global_i++ << endl ;
}

/*-----------------------------------------------------------------
global_i  = 10
global_i  = 11
-----------------------------------------------------------------*/
void callbyvalue::test_f1() {
  no_parameters_function() ;
  no_parameters_function() ;
}

/*-----------------------------------------------------------------
This function has no output, but has 2 inputs
-----------------------------------------------------------------*/
void callbyvalue::call_by_value_function(int coffee, int tea) {
  cout << "2 : coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
  int temp = coffee ;
  coffee = tea ;
  tea = temp ;
  global_i += 20 ;
  cout << "3 : coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
}

/*-----------------------------------------------------------------
1:  coffee = 10 tea = 20 global_i 10
2 : coffee = 10 tea = 20 global_i 10
3 : coffee = 20 tea = 10 global_i 30
4 : coffee = 10 tea = 20 global_i 30
-----------------------------------------------------------------*/
void callbyvalue::test_f2() {
  global_i = 10 ;
  int coffee = 10 ;
  int tea = 20 ;
  cout << "1:  coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
  call_by_value_function(coffee,tea) ;
  cout << "4 : coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
}

```

```
/*-----------------------------------------------------------------
This function has a output, but has 2 inputs
-----------------------------------------------------------------*/
int callbyvalue::call_by_value_and_return_function(int coffee, int tea) {
  cout << "6 : coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
  int temp = coffee ;
  coffee = tea ;
  tea = temp ;
  global_i += 20 ;
  cout << "7 : coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
  int x = coffee + tea ;
  return x ;
}

/*-----------------------------------------------------------------
global_i 10
5:  coffee = 10 tea = 20 global_i 24
6 : coffee = 10 tea = 20 global_i 10
7 : coffee = 20 tea = 10 global_i 30
8 : coffee = 10 tea = 20 global_i 25
9 : x = 30
global_i 30
-----------------------------------------------------------------*/
void callbyvalue::test_f3() {
  global_i = 10 ;
  cout << "global_i " << global_i << endl ;
  {
    int coffee = 10 ;
    int tea = 20 ;
    int global_i = 24 ;
    cout << "5:  coffee = " << coffee << " tea = " << tea  << " global_i " << global_i++ << endl ;
    int x = call_by_value_and_return_function(coffee,tea) ;
    cout << "8 : coffee = " << coffee << " tea = " << tea  << " global_i " << global_i << endl ;
    cout << "9 : x = " << x << endl ;
  }
  cout << "global_i " << global_i << endl ;
}

//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: callbyvaluetest.cpp

On linux:
g++ callbyvalue.cpp callbyvaluetest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test callbyvalue object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "callbyvalue.h"


/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  callbyvalue a ;
  a.test_f1() ;
  a.test_f2() ;
  a.test_f3() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF
```

VASUDEVAMURTHY

1.17 Introductions to pointers

Figure 1.27: Pointers

VASUDEVAMURTHY

Figure 1.28: Swaping contents of two variables

VASUDEVAMURTHY

is ``int* p;'' right or is ``int *p;'' right?

Both are "right" in the sense that both are valid C and C++ and both have exactly the same meaning.

As far as the language definitions and the compilers are concerned we could just as well say ``int*p;'' or ``int * p;''

A ``typical C programmer'' writes ``int *p;'' and explains it ``*p is what is the int'

A ``typical C++ programmer'' writes ``int* p;'' and explains it ``p is a pointer to an int''

In this course we use this notation:
We read from right to left

```
int x ; x is an integer
int* p ; p is a pointer to integer
int** p ; p is a pointer, pointing to another pointer, pointing to integer

int& p; p is an alias to an integer object
```

Figure 1.29: int *p or int* p

VASUDEVAMURTHY

1.17.1 C++code for illustrating pointers

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: pointers.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has pointers class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef pointers_H
#define pointers_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of pointers class
-----------------------------------------------------------------*/
class pointers{
public:
  void p1();
  void swap_that_does_not_swap();
  void swap() ;
  void why_pointer_to_pointer_exists() ;

private:
  void _p2(int* i);
  void _p3(int** i);
  void _p4(int*** i);
  void _call_by_value_function(int coffee, int tea);
  void _call_by_address_function(int* coffee, int* tea);
  void _f2(const char*** y);
  void _f1(const char** x);
};

#endif
//EOF
```

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: pointers.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "pointers.h"

/*-----------------------------------------------------------------
Definition of routines of pointers class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void pointers::_p4(int*** i) {
  cout << "in p4: " << "address of address of address of  i = " << i;
  cout << " and i = " << *(*(*(i))) << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void pointers::_p3(int** i) {
  cout << "in p3: " << "address of address of i = " << i;
  cout << " and i = " << *(*i )<< endl ;
  int*** aaai = &i ;
  _p4(aaai) ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void pointers::_p2(int* i) {
  cout << "in p2: " << "address of i = " << i;
  cout << " and i = " << *i << endl ;
  int** aai = &i ;
  _p3(aai) ;
}

/*-----------------------------------------------------------------
in p1: i = 189
in p2: address of i = 0034F04C and i = 189
```

```
in p3: address of address of i = 0034F034 and i = 189
in p4: address of address of address of  i = 0034F014 and i = 189
----------------------------------------------------------------*/
void pointers::p1() {
  int i = 189 ;
  cout << "in p1: " << "i = " << i << endl ;
  int* ai = &i ;
  _p2(ai) ;
}

/*----------------------------------------------------------------
This function has no output, but has 2 inputs
----------------------------------------------------------------*/
void pointers::_call_by_value_function(int coffee, int tea) {
  cout << "2 : coffee = " << coffee << " tea = " << tea  << endl ;
  int temp = coffee ;
  coffee = tea ;
  tea = temp ;
  cout << "3 : coffee = " << coffee << " tea = " << tea  << endl ;
}

/*----------------------------------------------------------------
1:  coffee = 10 tea = 20
2 : coffee = 10 tea = 20
3 : coffee = 20 tea = 10
4 : coffee = 10 tea = 20
----------------------------------------------------------------*/
void pointers::swap_that_does_not_swap() {
  int coffee = 10 ;
  int tea = 20 ;
  cout << "1:  coffee = " << coffee << " tea = " << tea  << endl ;
  _call_by_value_function(coffee,tea) ;
  cout << "4 : coffee = " << coffee << " tea = " << tea  << endl ;
}

/*----------------------------------------------------------------
This function has no output, but has 2 inputs
----------------------------------------------------------------*/
void pointers::_call_by_address_function(int* coffee, int* tea) {
  cout << "2 : coffee = " << *(coffee) << " tea = " << *(tea)  << endl ;
  int temp = *coffee ;
  *coffee = *tea ;
  *tea = temp ;
  cout << "3 : coffee = " << *(coffee) << " tea = " << *(tea)  << endl ;
}

/*----------------------------------------------------------------
1:  coffee = 10 tea = 20
2 : coffee = 10 tea = 20
```

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: pointerstest.cpp

On linux:
g++ pointers.cpp pointerstest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test pointers object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "pointers.h"


/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  pointers a ;
  a.p1() ;
  a.swap_that_does_not_swap();
  a.swap() ;
  a.why_pointer_to_pointer_exists();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF
```

1.18 Understanding const

```
char* p = "jag"; //Non constant pointer, Non constant data
const char* p = "jag" ; //Non constant pointer, constant data
char* const p = "jag" ; //constant pointer,  non constant data
const char* const p = "jag" ; //constant pointer,  constant data
```

What's pointed to is constant	Pointer is constant
char *	p = "jag"
const char *	p = "jag"
char *	const p = "jag"
const char *	const p = "jag"

Use const whenever possible

Figure 1.30: Use **const** whenever possible

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: const.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
integer test
-----------------------------------------------------------------*/
static void int_test() {
  int x = 4 ;
  x = 10 ;
  //const int y = 2 ;
  //y = 89 ;
  //FAILS: error C3892: 'y' : you cannot assign to a variable that is const
}

/*-----------------------------------------------------------------
Generic print routine
-----------------------------------------------------------------*/
static void print(const char* title, const char* s) {
  cout << title << endl ;
  int l = strlen(s) ;
  for (int i = 0; i < l; i++) {
    cout << s[i] ;
  }
  cout << endl ;
}

/*-----------------------------------------------------------------
Non constant pointer Non constant data
-----------------------------------------------------------------*/
static void ncp_ncd() {
  char* p = new char[4] ;
  strcpy(p,"NPU") ;
  print("ncp_ncd BEFORE: ", p) ;
  p[0] = 'L' ;
  print("ncp_ncd AFTER: ", p) ;
  delete [] p ;
}

/*-----------------------------------------------------------------
Non constant pointer Constant data
-----------------------------------------------------------------*/
static void ncp_cd() {
  const char* p = "MIT" ;
  cout << "p =  " << p << endl ;
  //p[0] = 'N' ;
  //error C3892: 'p' : you cannot assign to a variable that is const
  const char* q = "UCSC" ;
  p = q ;
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Constant pointer Non constant data
-----------------------------------------------------------------*/
static void cp_ncd() {
  char* const p = new char[4] ;
  strcpy(p,"NPU") ;
  print("cp_ncd BEFORE: ", p) ;
  p[0] = 'L' ;
  print("cp_ncd AFTER: ", p) ;

  char* const p1 = new char[4] ;
  strcpy(p1,"MIT") ;
  print("cp_ncd BEFORE2: ", p1) ;
  //p = p1 ;
  // error C3892: 'p' : you cannot assign to a variable that is const
  delete [] p ;
  delete [] p1 ;
}

/*-----------------------------------------------------------------
Constant pointer Constant data
-----------------------------------------------------------------*/
static void cp_cd() {
  const char* const  p = "MIT" ;
  //p[0] = 'K' ;
  //error C3892: 'p' : you cannot assign to a variable that is const

  const char* q = "UCSC" ;
  //p = q ;
  //error C3892: 'p' : you cannot assign to a variable that is const
}

/*-----------------------------------------------------------------
Main
-----------------------------------------------------------------*/
int main() {
  int_test();
  cout << "_____________________________________" << endl ;
  ncp_ncd() ;
  cout << "_____________________________________" << endl ;
  ncp_cd() ;
  cout << "_____________________________________" << endl ;
  cp_ncd();
  cout << "_____________________________________" << endl ;
  cp_cd() ;
  cout << "_____________________________________" << endl ;
  return 0 ;
}

```

VASUDEVAMURTHY

1.19 Introductions to arrays

```
int a[5] ;  //a is an array of 5 int elements starting from 0 to 4
```

'size', in this case 5, must be known before. CANNOT be changed

a[0]	a[1]	a[2]	a[3]	a[4]

We should carry size of array, 5 in this case.
There is no way of getting size, after declaration
You can do a[23] and get crash/unpredictable results

2000 2004 2008 2012 2016 2020

a[0]	a[1]	a[2]	a[3]	a[4]

a is an array
pa is variable

```
int* pa = a;     //pa is a pointer to the a[0]  ---> int* pa = &a[0] ;
int x = *(pa);   //Gives the content of a[0]
```

a[i] = *(a + i)

Note that pointers are not integer

```
pa++ ; //Not 2001, but 2004 (size of integer is 4)
int x = *(pa); //Gives the content of a[1]
pa = pa+2 ; //Goes to (2004 + (2 *4) = 2012
x = *(pa) ; //Gives the content of a[3]
```

```
&(a[0)] = pa = 2000
&(a[1]) = 2000 + ( 1 * 4) = 2004 = pa++
&(a[2]) = 2000 + ( 2 * 4) = 2008 = pa+=2
&(a[3)] = 2000 + ( 3 * 4) = 2012 = pa+=3
&(a[4]) = 2000 + ( 4 & 4) = 2016 = pa+=4
```

you cannot do a++, but pa++

Figure 1.31: Arrays and pointers to array elements

VASUDEVAMURTHY

Array's are NEVER passed by value

Declaring array

```
int a[5]
int* pa = a ;      pointer to 0th position of a
int* pa = &(a[0]) ; //same as above
int b = a ; //NOT POSSIBLE
int* pa3 = &(a[3]); //Pointer to 3rd position of a
```

a is an array
pa is variable

```
static void int_array() {
    const int SIZE = 5 ;
    int a[SIZE] ;
    int total_size = sizeof(a) ; //Here it knows the size of array
    int array_size = sizeof(a)/sizeof(int) ; //WOW.#of elements in a
}
```

CASE 1 **PASSING ARRAY - CASE 1**

```
static void print_array( int b[] , int size) {
    int k = sizeof(b) ;
    //HERE IT DOES NOT KNOW. It thinks as a pointer and hence k = 4
    //That means you need to pass correct size all the time.
    for (int i = 0; i < size; i++) {
        cout << b[i] << " " ;
    }
    cout << endl ;
}
```

```
static void f1() {
    const int SIZE = 5 ;
    int a[SIZE] ;
    print_array(a,SIZE) ;
}
```

CASE 2 **PASSING ARRAY - CASE 2**

```
static void inc_array( int* b , int size, int incr_amount) {
    int k = sizeof(b) ;
    //HERE IT DOES NOT KNOW. It thinks as a pointer and hence k = 4
    //That means you need to pass correct size all the time.
    for (int i = 0; i < size; i++) {
        b[i] = b[i] + incr_amount;
    }
}
```

```
static void f2() {
    const int SIZE = 5 ;
    int a[SIZE] ;
    inc_array(a,SIZE,40) ;
}
```

Figure 1.32: Passing array as a parameter of a function

VASUDEVAMURTHY

1.19.1 C++code for illustrating array and pointers to array elements

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: intarray.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has intarray class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef intarray_H
#define intarray_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of intarray class
-----------------------------------------------------------------*/
class intarray{
public:
  void array_basic1();
  void array_basic2();
private:
  void _print_array(const char* title, int b[], int size);
  void _print_in_reverse_order(const char* title, int b[], int size);
  void _inc_array(int* b, int size, int incr_amount);
  void _dec_array(int b[], int size, int decr_amount);
};

#endif
//EOF


```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: intarray.cpp
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has enumeration class definition
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "intarray.h"

/*----------------------------------------------------------------
Definition of routines of intarray class
----------------------------------------------------------------*/

/*----------------------------------------------------------------
sizeof(a)= 20
sizeof(a)/sizeof(int) = 5
a != b
a == b
a[0]= 23 a[1]= 78 a[2]= 90 a[3]= -45
a[0]= 23 a[1]= 78 a[2]= 90 a[3]= -45
----------------------------------------------------------------*/
void intarray::array_basic1() {
  {
    int size = 5 ;
    //SIZE MUST BE KNOWN at COMPILATION TIME. MUST BE A CONSTANT
    //int a[size] ;
    //error C2057: expected constant expression
  }
  {
    const int size = 5 ;
    int a[size] ;
    int k = sizeof(a) ;
    cout << "sizeof(a)= " << k << endl ;
    int num_element = sizeof(a)/sizeof(int) ;
    cout << "sizeof(a)/sizeof(int) = " << num_element << endl ;
    for (int i = 0; i < size; ++i) {
      a[i] = i ;
    }
    int b[size] ;
    //YOU cannot make b = a
    //b = a ;
    //error C2106: '=' : left operand must be l-value
    for (int i = 0; i < size; ++i) {
      b[i] = a[i];
    }
    if (a == b) {
      cout << "a == b" << endl ;
    }else {
      cout << "a != b" << endl ;
    }
    bool eq = true ;
    for (int i = 0; i < size; ++i) {
      if (b[i] != a[i]) {
        eq = false ;
        break ;
      }
    }
    if (eq) {
      cout << "a == b" << endl ;
    }else {
      cout << "a != b" << endl ;
```

VASUDEVAMURTHY

```
    }
  }
  {
    //This shows how to initalize the array
    int a[] = {23,78,90,-45};
    int num_elements = sizeof(a)/sizeof(int) ;
    for (int i = 0; i < num_elements; ++i) {
      cout << " a[" << i << "]= "<< a[i] ;
    }
    cout << endl ;
    int* p = a;
    for (int i = 0; i < num_elements; ++i) {
      cout << " a[" << i << "]= "<< *p++; ;
    }
    cout << endl ;
  }
  {
    //This shows how to initalize the array
    const char* a[] = { "Jan","feb","mar","april" };
    int num_elements = sizeof(a) / sizeof(const char*);
    for (int i = 0; i < num_elements; ++i) {
      cout << " a[" << i << "]= " << a[i];
    }
    cout << endl;
    const char** p = a;
    for (int i = 0; i < num_elements; ++i) {
      cout << " a[" << i << "]= " << *p++;;
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
print the content of array 'b' that has size 'size'
Note that
1. array is passed by address
2. size is passed by value
-----------------------------------------------------------------*/
void intarray::_print_array(const char* title, int b[], int size) {
  int k = sizeof(b) ;
  cout << "k = " << k << endl ;
  //HERE IT DOES NOT KNOW. It thinks as a pointer and hence k = 4
  //That means you need to pass correct size all the time.
  cout << title << ":" ;
  for (int i = 0; i < size; i++) {
    cout << b[i] << " " ;
  }
  cout << endl ;
}

/*-----------------------------------------------------------------
address of b[9]= 0032EBBC content of b[9]= 14
-----------------------------------------------------------------*/
void intarray::_print_in_reverse_order(const char* title, int b[], int size) {
  int k = sizeof(b) ;
  cout << "k = " << k << endl ;
  //HERE IT DOES NOT KNOW. It thinks as a pointer and hence k = 4
  //That means you need to pass correct size all the time.
  cout << title << ":" << endl;
  b = b + size - 1; //note that size is (size * 'size of integer')
  for (int i = size-1; i >= 0; i--) {
    cout << "address of b[" << i << "]= " << b << " content of b[" << i << "]= " << *b << endl ;
    b-- ; //b is not decremented by 1, Will decrement b by 'size of integer' which is 4
  }
}
```

```
/*-----------------------------------------------------------------
Increment the content of array 'b' that has size 'size' by 'incr_amount'
Note that
1. array is passed by address
2. size is passed by value
3. incr_amount is passed by value
-----------------------------------------------------------------*/
void intarray::_inc_array(int* b, int size, int incr_amount) {
  int k = sizeof(b) ;
  cout << "k = " << k << endl ;
  //HERE IT DOES NOT KNOW. It thinks as a pointer and hence k = 4
  //That means you need to pass correct size all the time.
  for (int i = 0; i < size; i++) {
    b[i] = b[i] + incr_amount;
  }
}

/*-----------------------------------------------------------------
decrement the content of array 'b' that has size 'size' by 'decr_amount'
Note that
1. array is passed by address
2. size is passed by value
3. decr_amount is passed by value
4. b++, which increment by 'size of integer'
-----------------------------------------------------------------*/
void intarray::_dec_array(int b[], int size, int decr_amount) {
  for (int i = 0; i < size; i++) {
    cout << "address of b[" << i << "]= " << b << " content of b[" << i << "]= " << *b << endl ;
    *b = (*b) - decr_amount ;
    b++ ; //b is not incremented by 1, Will increment b by 'size of integer' which is 4
  }
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void intarray::array_basic2() {
  const int SIZE = 10 ;
  int a[SIZE] ;
  int total_size = sizeof(a) ; //Here it knows the size of array
  cout << "total size = " << total_size << endl ;
  int array_size = sizeof(a)/sizeof(int) ; //WOW. number of element in a
  cout << "array size = " << array_size << endl ;
  for (int i = 0; i < SIZE; i++) {
    a[i] = i + 5 ;
  }
  _print_array("int_array", a,SIZE) ;
  _inc_array(a,SIZE,22) ;
  _print_array("After incrementing by 22", a,SIZE) ;
  _dec_array(a,SIZE,22) ;
  _print_array("After decrementing by 22", a,SIZE) ;
  _print_in_reverse_order("After reversing",a,SIZE) ;
}


//EOF
```

VASUDEVA MURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: intarraytest.cpp

On linux:
g++ intarray.cpp intarraytest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test intarray object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "intarray.h"


/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  intarray a ;
  a.array_basic1();
  a.array_basic2();
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF


```

VASUDEVAMURTHY

1.20 Basic stream output and stream input

Figure 1.33: Why switch to something new?

VASUDEVAMURTHY

How to write and read?

```
#include <iostream>
#include <fstream>

#include <iomanip>      // std::setprecision
using namespace std;
```

class ostream — **WRITE TO SCREEN**

ostream class can be used only with the instance of the class.

```
ostream cout ;
```

This object is already instantiated at global scope under namespace std

```
ostream cout ;
ostream cerr ;
ostream clog ;

int i ;
dog d ;
char ch ;

cout << i << " I am here " << d << ch << endl ;
```

class ofstream — **WRITE TO A FILE**

```
ofstream out("c:\\jag\c\\data.dat") ;
out << i << " I am here " << d << ch << endl ;
```

READ FROM KEYBOARD

class istream

```
istream cin;
int i ; cin >> i ; //read int
char ch; cin >> ch ; //read
```

READ FROM A FILE

class ifstream

```
ifstream in("c:/jag/myoutput/in.dat") ;
int i ; in << i ; //Read integer fron file
```

Figure 1.34: How to write and read data?

VASUDEVAMURTHY

1.20.1 C++code for illustrating stream output

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef output_H
#define output_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of output class
-----------------------------------------------------------------*/
class output{
public:
  void default_output_of_6() ;
  void how_to_format_real_numbers() ;
  void how_to_display_integers_in_proper_base() ;
  void how_to_set_width_and_fill_character() ;
  void file_default_output_of_6() ;
  void how_to_alternate_screen_and_file_output();
private:

};

#endif
//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: output.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "output.h"

/*-----------------------------------------------------------------
Definition of routines of output class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Note that default precision for float or double is 6
double d = 56.00009 is printed as d = 56.0001 (TOTAL WIDTH IS 6)

Note that float f0 = 56 and float d0 = 89 is printed with no decimal point
-----------------------------------------------
n = 65 f = 56.87  d = 56.0001  ch = a b = 1
f0 = 56 d0 = 89
----------------------------------------
-----------------------------------------------------------------*/
void output::default_output_of_6() {
  int n = 65 ;
  float f0 = 56;
  float f = 56.87f ;
  float d0 = 89 ;
  double d = 56.00009 ;
  char ch = 'a' ;
  bool b = true ;

  cout << "n = " << n << " " ;
  cout << "f = " << f << " " <<  " d = " << d ; ////Total width (number.number) is 6
  cout << "  ch = " << ch << " b = " << b << endl ;
  cout << "f0 = " << f0 << " d0 = " << d0 << endl ;
  cout << "----------------------------------------" << endl ;
}

/*-----------------------------------------------------------------
f = 56.86999893  d = 56.00009000
----------------------------------------
f = 57.  d = 56. f0 = 56. d0 = 89.
----------------------------------------
f = 56.87  d = 56.00 f0 = 56.00 d0 = 89.00
----------------------------------------
f = 56.87  d = 56 f0 = 56 d0 = 89
----------------------------------------
-----------------------------------------------------------------*/
void output::how_to_format_real_numbers() {
  float f0 = 56;
  float f = 56.87f ;
  float d0 = 89 ;
  double d = 56.00009 ;

  cout.precision(10) ;//Total width (number.number) is 10
  cout.setf(ios::showpoint) ;
  cout << "f = " << f << " " <<  " d = " << d ;
  cout << "\n----------------------------------------" << endl ;

  cout.precision(2) ; //Total width (number.number) is 2
```

```
  cout.setf(ios::showpoint) ;
  cout << "f = " << f << " " <<  " d = " << d ;
  cout << " f0 = " << f0 << " d0 = " << d0 << endl ;
  cout << "---------------------------------------" << endl ;

  cout.precision(4) ; //Total width (number.number) is 4
  cout.setf(ios::showpoint) ;
  cout << "f = " << f << " " <<  " d = " << d ;
  cout << " f0 = " << f0 << " d0 = " << d0 << endl ;
  cout << "---------------------------------------" << endl ;

  cout.unsetf(ios::showpoint) ;
  cout << "f = " << f << " " <<  " d = " << d ;
  cout << " f0 = " << f0 << " d0 = " << d0 << endl ;
  cout << "---------------------------------------" << endl ;
}

/*-----------------------------------------------------------------
In Decimal x = 65
In Octal x = 0101
In Hex x = 0x41
-----------------------------------------------------------------*/
void output::how_to_display_integers_in_proper_base() {
  int x = 65 ;
  cout.setf(ios::showbase) ;

  cout << "In Decimal x = " ;
  cout <<  x << endl ;

  cout << "In Octal x = " ;
  cout << oct << x << endl ;

  cout << "In Hex x = " ;
  cout << hex << x << endl ;
  cout << "---------------------------------------" << endl ;
  cout.unsetf(ios::showbase) ;
  cout << dec ; //Set the state back to decimal
}

/*-----------------------------------------------------------------
65
89
00065
***89
80
-----------------------------------------------------------------*/
void output::how_to_set_width_and_fill_character() {
  int x = 65 ;
  int y = 89 ;
  cout << setw(5) << x << endl ;
  cout << setw(5) << y << endl ;

  cout << setw(5) << setfill('0') << x << endl ;
  cout << setw(5) << setfill('*') << y << endl ;

  int z = 80 ;
  cout << z << endl ;
}

/*-----------------------------------------------------------------
Note that default precision for float or double is 6
double d = 56.00009 is printed as d = 56.0001 (TOTAL WIDTH IS 6)

Note that float f0 = 56 and float d0 = 89 is printed with no decimal point
-----------------------------------------------
n = 65 f = 56.87  d = 56.0001  ch = a b = 1
```


VASUDEVAMURTHY

```
f0 = 56 d0 = 89
----------------------------------------
-----------------------------------------------------------------*/
void output::file_default_output_of_6() {
#ifdef WIN32
  ofstream out("c:\\jag\\myoutput\\junk.dat") ;
#else
  ofstream out("c:/jag/myoutput/junk.dat") ;
#endif
  if (!out) {
    cout << "Cannot open junk.dat for writing" << endl ;
  }
  int n = 65 ;
  float f0 = 56;
  float f = 56.87f ;
  float d0 = 89 ;
  double d = 56.00009 ;
  char ch = 'a' ;
  bool b = true ;

  out << "n = " << n << " " ;
  out << "f = " << f << " " <<  " d = " << d ; ////Total width (number.number) is 6
  out << "  ch = " << ch << " b = " << b << endl ;
  out << "f0 = " << f0 << " d0 = " << d0 << endl ;
  out << "----------------------------------------" << endl ;
  out.flush() ;
  out.close() ;
#ifdef WIN32
  cout << "Answer is in: c:\\jag\\myoutput\\junk.dat\n" ;
#else
  cout << "Answer is in: c:/jag/myoutput/junk.dat\n" ;
#endif
}

/*-----------------------------------------------------------------
Writing to screen or to file
-----------------------------------------------------------------*/
void output::how_to_alternate_screen_and_file_output() {
#ifdef WIN32
  ofstream fout("c:\\jag\\myoutput\\junk.dat") ;
#else
  ofstream fout("c:/jag/myoutput/junk.dat") ;
#endif
  if (!fout) {
    cout << "Cannot open junk.dat for writing" << endl ;
  }
  for (int i = 0 ; i < 10; ++i) {
    ostream* out = &cout ; //Even numbers on screen
    if (i%2) {
      out = &fout ; //Odd numbers in file
    }
    *out << "Writing number " << i << endl ;
  }
  fout.close() ;
#ifdef WIN32
  cout << "Answer is in: c:\\jag\\myoutput\\junk.dat\n" ;
#else
  cout << "Answer is in: c:/jag/myoutput/junk.dat\n" ;
#endif
}
//EOF
```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: outputtest.cpp

On linux:
g++ output.cpp outputtest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test output object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "output.h"

/*----------------------------------------------------------------

----------------------------------------------------------------*/
void test1() {
  output o ;
  o.default_output_of_6() ;
  o.how_to_format_real_numbers() ;
  o.how_to_display_integers_in_proper_base() ;
  o.how_to_set_width_and_fill_character() ;
  o.file_default_output_of_6();
  o.how_to_alternate_screen_and_file_output();
}

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  test1() ;
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

1.20.2 C++code for illustrating stream input

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef input_H
#define input_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of input class
-----------------------------------------------------------------*/
class input{
public:
  void basic() ;
  void how_to_enter_in_loop();
  void how_to_check_error() ;
  void how_to_read_reliably();
  void how_to_read_from_file() ;
private:

};

#endif
//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: input.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "input.h"

/*-----------------------------------------------------------------
Definition of routines of input class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
You can space before or after.
If U make error problem
-----------------------------------------------------------------*/
void input::basic() {
  {
    cout << "Enter a number: " ;
    int x ;
    cin >> x ;
    cout << "You entered:" << x << endl ;
  }

  {
    cout << "Enter a character: " ;
    char x ;
    cin >> x ;
    cout << "You entered:" << x << endl ;
  }

  {
    cout << "Enter a float: " ;
    float x ;
    cin >> x ;
    cout << "You entered:" << x << endl ;
  }

  {
    cout << "Enter a double: " ;
    double x ;
    cin >> x ;
    cout << "You entered:" << x << endl ;
  }
}

/*-----------------------------------------------------------------
enter a number: 8
You entered:8
enter a number:   78
You entered:78
enter a number: 678
You entered:678
enter a number: 6780
You entered:6780
enter a number: ^Z


WINDOWS: <CONTROL>Z
UNIX : <CONTROL>D
```

VASUDEVAMURTHY

```
----------------------------------------------------------------*/
void input::how_to_enter_in_loop() {
  bool isend = false ;
  do {
    cout << "enter a number: " ;
    int x ;
    cin >> x ;
    isend = cin.eof() ;
    if (!isend) {
      cout << "You entered:" << x << endl ;
    }
  }while (!isend) ;
}

/*----------------------------------------------------------------
<CONTROL>Z will set eof to true

enter a number: y
eof() = false
good = false
bad = false
fail = true
----------------------------------------------------------------*/
void input::how_to_check_error() {
  cout << "enter a number: " ;
  int x ;
  cin >> x ;

  cout << boolalpha ; //displays boolean as true or false

  cout << "eof() = " << (cin.eof()) << endl ;
  cout << "good = " << (cin.good()) << endl ;
  cout << "bad = " << (cin.bad()) << endl ;
  cout << "fail = " << (cin.fail()) << endl ;

  cout.unsetf(ios::boolalpha) ; //unset boolalpha
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
void input::how_to_read_reliably() {
  const INT MAX = 1024 ;
  bool error = false ;
  do {
    error = false ;
    cout << "Enter integer: " ;
    char buffer[MAX] ;
    cin.getline(buffer,MAX) ;
    int i = 0 ;
    int num = 0 ;
    char ch = 'a' ;
    do {
      ch = buffer[i++] ;
      if (ch != '\0') {
        if (ch >= '0' && ch <= '9') {
          num = num*10 + (ch - '0') ;
        }else {
          error = true ;
        }
      }
    }while (!error && (ch != '\0')) ;
    if (error) {
      cout << "You entered garbage:" << buffer << endl ;
    }else {
      cout << "Correct Input:" << num << endl ;
```

```
    }
  }while (error) ;
}


/*-----------------------------------------------------------------
in.dat
34 a
67 h
78 j
90 k
89 k
-----------------------------------------------------------------*/
void input::how_to_read_from_file() {
#ifdef WIN32
  ifstream in("c:\\jag\\myoutput\\in.dat") ;
#else
  ifstream in("c:/jag/myoutput/in.dat") ;
}
#endif
  if (!in) {
    cout << "Cannot open in.dat for reading" << endl ;
  }
  do{
    int n = -1;
    char ch = 'z' ;
    in >> n >> ch ;
    if (!in.eof()) {
      cout << "Read: " << n << " " << ch << endl ;
    }else {
      break ;
    }
  }while(1) ;
}

//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: inputtest.cpp

On linux:
g++ input.cpp inputtest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test input object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "input.h"

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void test1() {
  input i ;
  i.basic() ;
  i.how_to_enter_in_loop();
  i.how_to_check_error() ;
  i.how_to_read_reliably() ;
  i.how_to_read_from_file() ;

}

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  test1() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

1.21 Problem set

Problem 1.21.1. Write a *class* called **temperature** that performs the following tasks:

- Write a public function that converts Fahrenheit temperature to Centigrade temperature using the equation

$$C = (5/9) * (F - 32)$$

- Write a public function that converts Centigrade temperature to Fahrenheit temperature using the equation

$$F = ((9/5) * T) + 32;$$

- Test your program from a Fahrenheit temperature of -50 degrees to +50 degrees, in a step of five degrees. Convert the centigrade answer you got to Fahrenheit temperature. At what temperature the Fahrenheit temperature matches with the Centigrade temperature.

Problem 1.21.2. Write a *class* called **p1** as decribed in Fig 1.35.

VASUDEVAMURTHY

Figure 1.35: Various functions of class P1

VASUDEVAMURTHY

class p1

p1test.cpp
a.riceOnChessBoard(64);

```
Rice on Square 1 = 1
Rice on Square 2 = 2
Rice on Square 3 = 4
Rice on Square 4 = 8
Rice on Square 5 = 16
Rice on Square 6 = 32
Rice on Square 7 = 64
Rice on Square 8 = 128
Rice on Square 9 = 256
Rice on Square 10 = 512
Rice on Square 11 = 1024
Rice on Square 12 = 2048
Rice on Square 13 = 4096
Rice on Square 14 = 8192
Rice on Square 15 = 16384
Rice on Square 16 = 32768
Rice on Square 17 = 65536
Rice on Square 18 = 131072
Rice on Square 19 = 262144
Rice on Square 20 = 524288
Rice on Square 21 = 1048576
Rice on Square 22 = 2097152
Rice on Square 23 = 4194304
Rice on Square 24 = 8388608
Rice on Square 25 = 16777216
Rice on Square 26 = 33554432
Rice on Square 27 = 67108864
Rice on Square 28 = 134217728
Rice on Square 29 = 268435456
Rice on Square 30 = 536870912
Rice on Square 31 = 1073741824
Rice on Square 32 = 2147483648
Rice on Square 33 = 4294967296
Rice on Square 34 = 8589934592
Rice on Square 35 = 17179869184
Rice on Square 36 = 34359738368
Rice on Square 37 = 68719476736
Rice on Square 38 = 137438953472
Rice on Square 39 = 274877906944
Rice on Square 40 = 549755813888
Rice on Square 41 = 1099511627776
Rice on Square 42 = 2199023255552
Rice on Square 43 = 4398046511104
Rice on Square 44 = 8796093022208
Rice on Square 45 = 17592186044416
Rice on Square 46 = 35184372088832
Rice on Square 47 = 70368744177664
Rice on Square 48 = 140737488355328
Rice on Square 49 = 281474976710656
Rice on Square 50 = 562949953421312
Rice on Square 51 = 1125899906842624
Rice on Square 52 = 2251799813685248
Rice on Square 53 = 4503599627370496
Rice on Square 54 = 9007199254740992
Rice on Square 55 = 18014398509481984
Rice on Square 56 = 36028797018963968
Rice on Square 57 = 72057594037927936
Rice on Square 58 = 144115188075855872
Rice on Square 59 = 288230376151711744
Rice on Square 60 = 576460752303423488
Rice on Square 61 = 1152921504606846976
Rice on Square 62 = 2305843009213693952
Rice on Square 63 = 4611686018427387904
Rice on Square 64 = 9223372036854775808
Total rice = 18446744073709551615
Total rice must  be = 18446744073709551615
```

Figure 1.36: Rice on a chess board of class P1

VASUDEVAMURTHY

Problem 1.21.3. Write a *class* called **collatz** for the **Collatz conjecture** described in Fig 1.37.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Collatz conjecture

Consider the following operation on an arbitrary positive integer:

- If the number is even, divide it by two.
- If the number is odd, triple it and add one.

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n+1 & \text{if } n \text{ is add} \end{cases}$$

For instance, starting with n = 6, one gets the sequence 6, 3, 10, 5, 16, 8, 4, 2, 1.

Run your program for n =11 and n = 27

Figure 1.37: Collatz conjecture

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
file: collatztest.cpp

On linux:
g++ collatz.cpp collatztest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test collatz object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "collatz.h"

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  collatz c;
  const int MAX = 1000;
  int a[MAX];
  for (int i = 0; i < 100; ++i) {
    int s = c.run(i, a, MAX);
    c.print(s, a);
  }
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
  testbed();
  return 0;
}

//EOF
```

Problem 1.21.4. Write a *class* called **caesar** for the **Caesar cipher** described in Fig 1.38. You must write two files **caesar.h** and **caesar.cpp** and test your class using the provided file **caesartest.cpp** Your program must work for all the cases provided in **caesartest.cpp**. You cannot change anything in the file **caesartest.cpp**. **email** only **caesar.h** and **caesar.cpp**. Also include screen shot of the program output as a pdf file.

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: caesartest.cpp

On linux:
g++ caesar.cpp caesartest.cpp
valgrind a.out
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test caesar object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "caesar.h"

/*----------------------------------------------------------------

----------------------------------------------------------------*/
void test1(int k, const char* s) {
  cout << "-------------------------------------------------------\n";
  if (k < 0) {
    cout << "Number of rotation cannot be negative\n";
  }
  else {
    char eans[100];
    char dans[100];
    caesar a;
    a.encrypt(k, s, eans);
    a.decrypt(k, eans, dans);
    cout << "Rotating by " << k << endl;
    cout << "original string  = " << s << endl;
    cout << "Encrypted string = " << eans << endl;
    cout << "Decrypted string = " << dans << endl;
    assert(strcmp(s, dans) == 0);
  }
  cout << "-------------------------------------------------------\n";
}

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void test2(const char* s) {
  test1(0, s); //No rotation
  test1(26, s); //No rotation
  test1(1, s); //One rotation
  test1(6, s); //six rotation
  test1(13, s);
  test1(-1, s);
  test1(45, s);
  test1(31, s);
  test1(30, s);
  test1(420, s);
}

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  test2("abcdefghijklmnopqrstuvwxyz");
  test2("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
  test2("Be sure to drink your Ovaltine3!");
  test2("Pack My Box With Five Dozens LIQUOR jugs");
  test2("I am going to fail YOU");
```

```
  test2("A Quick BROWN fox jumps over a lazy DOG");
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed();
  return 0;
}

//EOF
```

VASUDEVAMURTHY

Caesar cipher

In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.
For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who used it in his private correspondence

Example: With a shift of 13, k = 13,
original string = I am going to fail YOU
Encrypted string = V nz tbvat gb snvy LBH
Decrypted string = I am going to fail YOU

With k = 420,
original string = A Quick BROWN fox junps over a lazy DOG
Encrypted string = E Uymgo FVSAR jsb nyrtw sziv e pedc HSK
Decrypted string = A Quick BROWN fox junps over a lazy DOG

Your task is to write a class called:

```
class caesar{
private:                    Number of rotation
public:
  void encrypt(int k, const char* s, char* ans);
  void decrypt(int k, const char* s, char* ans);
};
```

You must write code in: caesar.h and caesar.cpp
and test your code using caesartest.cpp which is provided to you.
YOU CANNOT CHANGE ANYTHING IN caesartest.cpp.
E-mail caesar.h and caesar.cpp.

Figure 1.38: Caesar cipher

VASUDEVAMURTHY

Problem 1.21.5. Write a *class* called **reverse1**as explained in Figure 1.39

Figure 1.39: Reverse an unsigned long number

VASUDEVAMURTHY

Problem 1.21.6. Write a *class* called **luhn** for **validating a credit card** as described in Fig 1.40. You must write two files **luhn.h** and **luhn.cpp** and test your class using the provided file **luhntest.cpp** You cannot use an **array** to solve this problem. Your program must work for all the cases provided in **luhntest.cpp**. You cannot change anything in the file **luhntest.cpp**.

Validating Credit Card Numbers Using LUHN Algorithm

4 0 1 2 8 8 8 8 8 8 8 8 8 1 8 8 1

Step 1 - Starting with the check digit double the value of every other digit (right to left every 2nd digit)

4	0	1	2	8	8	8	8	8	8	8	8	1	8	8	1
x2		x2		x2		x2		x2		x2		x2		x2	
8		2		16		16		16		16		2		16	

Step 2 - If doubling of a number results in a two digits number, add up the digits to get a single digit number. This will results in eight single digit numbers.

4	0	1	2	8	8	8	8	8	8	8	8	1	8	8	1
x2		x2		x2		x2		x2		x2		x2		x2	
8		2		7		7		7		7		2		7	

Step 3 - Now add the un-doubled digits to the odd places

4	0	1	2	8	8	8	8	8	8	8	8	1	8	8	1
x2		x2		x2		x2		x2		x2		x2		x2	
8		2		7		7		7		7		2		7	
	0		2		8		8		8		8		8		1

Step 4 - Add up all the digits in this number

4	0	1	2	8	8	8	8	8	8	8	8	1	8	8	1
x2		x2		x2		x2		x2		x2		x2		x2	
8		2		7		7		7		7		2		7	
	0		2		8		8		8		8		8		1

8+0+2+2+7+8+7+8+7+8+7+8+2+8+7+1
= 90

If the final sum is divisible by 10, then the credit card number is valid. If it is not divisible by 10, the number is invalid.

Test Credit Card Account Numbers

Credit Card Type	Credit Card Number
American Express	378282246310005
American Express	371449635398431
American Express Corporate	378734493671000
Australian BankCard	5610591081018250
Diners Club	30569309025904
Diners Club	38520000023237
Discover	6011111111111117
Discover	6011000990139424
JCB	3530111333300000
JCB	3566002020360505
MasterCard	5555555555554444
MasterCard	5105105105105100
Visa	4111111111111111
Visa	4012888888881881

Figure 1.40: Validating a credit card number using Luhn algorithm

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
file: luhntest.cpp

On linux:
g++ luhn.cpp luhntest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test luhn object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "luhn.h"

/*----------------------------------------------------------------
https://www.paypalobjects.com/en_US/vhelp/paypalmanager_help/credit_card_numbers.htm
----------------------------------------------------------------*/
void test1() {
  const long long list[] = { 378282246310005, 371449635398431, 378734493671000, 5610591081018250, 30569309025904, 38520000023237, 6011111111111117, 6011000990139424, 3530111333300000, 5105105105105100, 4111111111111111, 4012888888881881 };
  const bool ans[] = {0,0,0,1,0,0,1,1,1,1,1,1} ;
  int s = sizeof(list) / sizeof(long long);
  int a = sizeof(ans)/sizeof(bool) ;
  assert(s == a) ;
  for (int i = 0; i < s; ++i) {
    luhn a;
    long long j = list[i];
    assert(a.check(j) == ans[i]) ;
  }
}

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  luhn a;
  assert(a.check(4012888888881881));
  assert(a.check(378282246310005) == false);
  assert(a.check(4012888888881880) == false);
  assert(a.check(4222222222222) == false);
  test1();
```

```
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed();
  return 0;
}

//EOF


```

```
CARD NUMBER =    4012888888881881 Num digits =  16 Sum =    90  Valid
CARD NUMBER =     378282246310005 Num digits =  15 Sum =  60  Invalid
CARD NUMBER =    4012888888881880 Num digits =  16 Sum =  89  Invalid
CARD NUMBER =      4222222222222 Num digits =  13 Sum =  40  Invalid
CARD NUMBER =     378282246310005 Num digits =  15 Sum =  60  Invalid
CARD NUMBER =     371449635398431 Num digits =  15 Sum =  80  Invalid
CARD NUMBER =     378734493671000 Num digits =  15 Sum =  60  Invalid
CARD NUMBER =    5610591081018250 Num digits =  16 Sum =  40  Valid
CARD NUMBER =      30569309025904 Num digits =  14 Sum =  50  Invalid
CARD NUMBER =      38520000023237 Num digits =  14 Sum =  40  Invalid
CARD NUMBER =    6011111111111117 Num digits =  16 Sum =  30  Valid
CARD NUMBER =    6011000990139424 Num digits =  16 Sum =  50  Valid
CARD NUMBER =    3530111333300000 Num digits =  16 Sum =  40  Valid
CARD NUMBER =    5105105105105100 Num digits =  16 Sum =  20  Valid
CARD NUMBER =    4111111111111111 Num digits =  16 Sum =  30  Valid
CARD NUMBER =    4012888888881881 Num digits =  16 Sum =  90  Valid
```

Problem 1.21.7. Write a *class* called **array** as decribed in Fig 1.41.

class array

1

Reverse a string
You change the given string

before Hello
after olleH

before even
after neve

before Pack my box with five dozen liquor jugs
after sguj rouqil nezod evif htiw xob ym kcaP

2

Convert string number to integer

String is untouched. You will return int value of string.
return 0 if string has any character other than 0 to 9

1986 integer value = 1986
19a7 integer value = 0
19868978 integer value = 19868978
""integer value = 0

3

Remove Duplicates

You change the given array itself.
This procedure will return final size

BEFORE DUP 5 2 9 3 7
AFTER DUP 5 2 9 3 7

BEFORE DUP 2 2
AFTER DUP 2

BEFORE DUP 58 26 91 26 70 70 91 58 58 58 66
AFTER DUP 58 26 91 70 66

email:
array.h
array.cpp
arraytest.cpp

Figure 1.41: Various functions of class array

VASUDEVAMURTHY

Problem 1.21.8. Write a *class* called **PerpetualCalendar** that can find the day of the events given in figure 1.42 using the chart **Perpetual Calendar**

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Use perpetual calender to find the day of the following events

October 12, 1493 - Christopher Columbus sights land in the Americas.
May 29, 1493 - Constantinople falls. End of Byzantine empire.
January 30, 1649 - King Charles I of England is executed,
July 4, 1776 - Declaration of independence officially adopted. First moder
July 14, 1789 - Storming of the Bastille. Turning point in the French Revo
June 18, 1815 - Napoleon defeated at Battle of Waterloo.
June 28, 1914 - Archduke Ferdinand assassinated.
October 25, 1917 - Start of Bolshevik rebellion.
June 26, 1919 - Treaty of Versailles signed
March 23, 1933 - Hitler becomes dictator of Germany.
August 6, 1845 - US drops atomic bomb on Hiroshima.
October 22, 1960 - Cuban missile crisis.
November 9, 1989 - Berlin wall falls.
May 2, 2011 - Osama bin Laden killed by US Seals
Adolf Hitler committed suicide by gunshot on 30 April 1945

Leap Day – Famous Birthdays

1960 – Richard Ramirez, American serial killer
1964 – Lyndon Byers, Canadian hockey player
1972 – Antonio Sabàto Jr, Italian-born actor
1976 – Ja Rule, American rapper and actor
1980 – Chris Conley, American musician and songwriter/composer
1896 – Morarji Desai, former Indian prime minister
1924 – Carlos Humberto Romero, former president of El Salvador

Figure 1.42: What day of the week this events fall?

VASUDEVAMURTHY

Perpetual Calendar

These charts enable you to find the day of the week in any year from 1755 through 2033.

Instructions

STEP 1

Find the year that you are interested in below and note the letter that follows it.

STEP 2

Find that letter below and note which number falls under the month you are looking for.

STEP 3

Use the calendar that bears the number you found in Step 2.

YEARS

	1789D	1824K	1859F	1894A	1929B	1964J	1999E
1755C	1790E	1825F	1860N	1895B	1930C	1965E	2000M
1756K	1791F	1826G	1861B	1896J	1931D	1966F	2001A
1757F	1792N	1827A	1862C	1897E	1932L	1967G	2002B
1758G	1793B	1828I	1863D	1898F	1933G	1968H	2003C
1759A	1794C	1829D	1864L	1899G	1934A	1969C	2004K
1760I	1795D	1830E	1865G	1900A	1935B	1970D	2005F
1761D	1796L	1831F	1866A	1901B	1936J	1971E	2006G
1762E	1797G	1832N	1867B	1902C	1937E	1972M	2007A
1763F	1798A	1833B	1868J	1903D	1938F	1973A	2008I
1764N	1799B	1834C	1869E	1904L	1939G	1974B	2009D
1765B	1800C	1835D	1870F	1905G	1940H	1975C	2010E
1766C	1801D	1836L	1871G	1906A	1941C	1976K	2011F
1767D	1802E	1837G	1872H	1907B	1942D	1977F	2012N
1768L	1803F	1838A	1873C	1908J	1943E	1978G	2013B
1769G	1804N	1839B	1874D	1909E	1944M	1979A	2014C
1770A	1805B	1840J	1875E	1910F	1945A	1980I	2015D
1771B	1806C	1841E	1876M	1911G	1946B	1981D	2016L
1772J	1807D	1842F	1877A	1912H	1947C	1982E	2017G
1773E	1808L	1843G	1878B	1913C	1948K	1983F	2018A
1774F	1809G	1844H	1879C	1914D	1949F	1984N	2019B
1775G	1810A	1845C	1880K	1915E	1950G	1985B	2020J
1776H	1811B	1846D	1881F	1916M	1951A	1986C	2021E
1777C	1812J	1847E	1882G	1917A	1952I	1987D	2022F
1778D	1813E	1848M	1883A	1918B	1953D	1988L	2023G
1779E	1814F	1849A	1884I	1919C	1954E	1989G	2024H
1780M	1815G	1850B	1885D	1920K	1955F	1990A	2025C
1781A	1816H	1851C	1886E	1921F	1956N	1991B	2026D
1782B	1817C	1852K	1887F	1922G	1957B	1992J	2027E
1783C	1818D	1853F	1888N	1923A	1958C	1993E	2028M
1784K	1819E	1854G	1889B	1924I	1959D	1994F	2029A
1785F	1820M	1855A	1890C	1925D	1960L	1995G	2030B
1786G	1821A	1856I	1891D	1926E	1961G	1996H	2031C
1787A	1822B	1857D	1892L	1927F	1962A	1997C	2032K
1788I	1823C	1858E	1893G	1928N	1963B	1998D	2033F

MONTHS

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
A	1	4	4	7	2	5	7	3	6	1	4	6
B	2	5	5	1	3	6	1	4	7	2	5	7
C	3	6	6	2	4	7	2	5	1	3	6	1
D	4	7	7	3	5	1	3	6	2	4	7	2
E	5	1	1	4	6	2	4	7	3	5	1	3
F	6	2	2	5	7	3	5	1	4	6	2	4
G	7	3	3	6	1	4	6	2	5	7	3	5
H	1	4	5	1	3	6	1	4	7	2	5	7
I	2	5	6	2	4	7	2	5	1	3	6	1
J	3	6	7	3	5	1	3	6	2	4	7	2
K	4	7	1	4	6	2	4	7	3	5	1	3
L	5	1	2	5	7	3	5	1	4	6	2	4
M	6	2	3	6	1	4	6	2	5	7	3	5
N	7	3	4	7	2	5	7	3	6	1	4	6

CALENDARS

	1	2	3	4	5	6	7
Monday	1						
Tuesday	2	1					
Wednesday	3	2	1				
Thursday	4	3	2	1			
Friday	5	4	3	2	1		
Saturday	6	5	4	3	2	1	
SUNDAY	7	6	5	4	3	2	1
Monday	8	7	6	5	4	3	2
Tuesday	9	8	7	6	5	4	3
Wednesday	10	9	8	7	6	5	4
Thursday	11	10	9	8	7	6	5
Friday	12	11	10	9	8	7	6
Saturday	13	12	11	10	9	8	7
SUNDAY	14	13	12	11	10	9	8
Monday	15	14	13	12	11	10	9
Tuesday	16	15	14	13	12	11	10
Wednesday	17	16	15	14	13	12	11
Thursday	18	17	16	15	14	13	12
Friday	19	18	17	16	15	14	13
Saturday	20	19	18	17	16	15	14
SUNDAY	21	20	19	18	17	16	15
Monday	22	21	20	19	18	17	16
Tuesday	23	22	21	20	19	18	17
Wednesday	24	23	22	21	20	19	18
Thursday	25	24	23	22	21	20	19
Friday	26	25	24	23	22	21	20
Saturday	27	26	25	24	23	22	21
SUNDAY	28	27	26	25	24	23	22
Monday	29	28	27	26	25	24	23
Tuesday	30	29	28	27	26	25	24
Wednesday	31	30	29	28	27	26	25
Thursday		31	30	29	28	27	26
Friday			31	30	29	28	27
Saturday				31	30	29	28
SUNDAY					31	30	29
Monday						31	30
Tuesday							31

VASUDEVAMURTHY

For clarity, the alphabet associated with the years 1755 through 2033, is repeated below.

1755 C	1756 K	1757 F	1758 G	1759 A	1760 I	1761 D	1762 E	1763 F
1764 N	1765 B	1766 C	1767 D	1768 L	1769 G	1770 A	1771 B	1772 J
1773 E	1774 F	1775 G	1776 H	1777 C	1778 D	1779 E	1780 M	1781 A
1782 B	1783 C	1784 K	1785 F	1786 G	1787 A	1788 I	1789 D	1790 E
1791 F	1792 N	1793 B	1794 C	1795 D	1796 K	1797 G	1798 A	1799 B
1800 C	1801 D	1802 E	1803 F	1804 N	1805 B	1806 C	1807 D	1808 L
1809 G	1810 A	1811 B	1812 J	1813 E	1814 F	1815 G	1816 H	1817 C
1818 D	1819 E	1820 M	1821 A	1822 B	1823 C	1824 K	1825 F	1826 G
1827 A	1828 I	1829 D	1830 E	1831 F	1832 N	1833 B	1834 C	1835 D
1836 L	1837 G	1838 A	1839 B	1840 J	1841 E	1842 F	1843 G	1844 H
1845 C	1846 D	1847 E	1848 M	1849 A	1850 B	1851 C	1852 K	1853 F
1854 G	1855 A	1856 I	1857 D	1858 E	1859 F	1860 N	1861 B	1862 C
1863 D	1864 L	1865 G	1866 A	1867 B	1868 J	1869 E	1870 F	1871 G
1872 H	1873 C	1874 D	1875 E	1876 M	1877 A	1878 B	1879 C	1880 K
1881 F	1882 G	1883 A	1884 I	1885 D	1886 E	1887 F	1888 N	1889 B
1890 C	1891 D	1892 L	1893 G	1894 A	1895 B	1896 J	1897 E	1898 F
1899 G	1900 A	1901 B	1902 C	1903 D	1904 L	1905 G	1906 A	1907 B
1908 J	1909 E	1910 F	1911 G	1912 H	1913 C	1914 D	1915 E	1916 M
1917 A	1918 B	1919 C	1920 K	1921 F	1922 G	1923 A	1924 I	1925 D
1926 E	1927 F	1928 N	1929 B	1930 C	1931 D	1932 L	1933 G	1934 A
1935 B	1936 J	1937 E	1938 F	1939 G	1940 H	1941 C	1942 D	1943 E
1944 M	1945 A	1946 B	1947 C	1948 K	1949 F	1950 G	1951 A	1952 I
1953 D	1954 E	1955 F	1956 N	1957 B	1958 C	1959 D	1960 L	1961 G
1962 A	1963 B	1964 J	1965 E	1966 F	1967 G	1968 H	1969 C	1970 D
1971 E	1972 M	1973 A	1974 B	1975 C	1976 K	1977 F	1978 G	1979 A
1980 I	1981 D	1982 E	1983 F	1984 N	1985 B	1986 C	1987 D	1988 L
1989 G	1990 A	1991 B	1992 J	1993 E	1994 F	1995 G	1996 H	1997 C
1998 D	1999 E	2000 M	2001 A	2002 B	2003 C	2004 K	2005 F	2006 G
2007 A	2008 I	2009 D	2010 E	2011 F	2012 N	2013 B	2014 C	2015 D
2016 L	2017 G	2018 A	2019 B	2020 J	2021 E	2022 F	2023 G	2024 H
2025 C	2026 D	2027 E	2028 M	2029 A	2030 B	2031 C	2032 K	2033 F

VASUDEVAMURTHY

Problem 1.21.9. Write a *class* called **ninetynine** for **printing a poem** as described in Fig 1.43. You must write three files **ninetynine.h** and **ninetynine.cpp** and test your class using the file **ninetyninetest.cpp** Also include screen shot of the program output as a pdf file.

Figure 1.43: Automatically print the lyrics of a famous song

VASUDEVAMURTHY

VASUDEVAMURTHY

Chapter 2

C Strings

2.1 Introduction

2.2 Character manpulation

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: character.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has character class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef character_H
#define character_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of character class
-----------------------------------------------------------------*/
class character{
public:
  static bool isdigit(const char s);
  static bool isalpha(const char s);
  static bool isalnum(const char s);
  static bool is_ascii(const char s);
  static bool isupper(const char s);
  static bool islower(const char s);
  static char toupper(char s);
  static char tolower(char s);
private:

};

#endif
//EOF


```

VASUDEVAMURTHY

isalnum	test for alphanumeric character
isalpha	test for alphabetic character
isblank	test for blank character (new in C99)
iscntrl	test for control character
isdigit	test for digit. Not locale-specific.
isgraph	test for graphic character, excluding the space character.
islower	test for lowercase character
isprint	test for printable character, including the space character.
ispunct	test for punctuation character
isspace	test for any whitespace character
isupper	test for uppercase character
isxdigit	test for hexadecimal digit. Not locale-specific.

tolower	convert character to lowercase
toupper	convert character to uppercase

ctype.h

```
int isalnum(int c);
     isalpha(c) or isdigit(c)
int isalpha(int c);
     isupper(c) or islower(c)
int iscntrl(int c);
     is control character. In ASCII, control characters are 0x00 (NUL) to 0x1F (US), and 0x7F (DEL)
int isdigit(int c);
     is decimal digit
int isgraph(int c);
     is printing character other than space
int islower(int c);
     is lower-case letter
int isprint(int c);
     is printing character (including space). In ASCII, printing characters are 0x20 (' ') to 0x7E ('~')
int ispunct(int c);
     is printing character other than space, letter, digit
int isspace(int c);
     is space, formfeed, newline, carriage return, tab, vertical tab
int isupper(int c);
     is upper-case letter
int isxdigit(int c);
     is hexadecimal digit
int tolower(int c);
     return lower-case equivalent
int toupper(int c);
     return upper-case equivalent
```

Figure 2.1: Contents of **ctype.h**

VASI VASUDEVAMURTHY

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: character.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "character.h"

/*-----------------------------------------------------------------
Definition of routines of character class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Check if s is a digit
-----------------------------------------------------------------*/
bool character::isdigit(const char s) {
  return (s >= '0' && s <= '9') ? true : false ;
}

/*-----------------------------------------------------------------
Check if s is a character
-----------------------------------------------------------------*/
bool character::isalpha(const char s) {
  return ((s >= 'a' && s <= 'z') || (s >= 'A' && s <= 'Z')) ? true : false ;
}

/*-----------------------------------------------------------------
Check if character is a digit or a alphabetic character
-----------------------------------------------------------------*/
bool character::isalnum(const char s) {
  return  (isdigit(s) || isalpha(s)) ? true : false ;
}

/*-----------------------------------------------------------------
Check if character s is ascii
-----------------------------------------------------------------*/
bool character::is_ascii(const char s) {
  unsigned short u = s ;
  return (u >= 0 && u <=127) ? true : false ;
}

/*-----------------------------------------------------------------
Check if character is uppercase letter
-----------------------------------------------------------------*/
bool character::isupper(const char s) {
  return (s >= 'A' && s <= 'Z') ? true : false ;
}

/*-----------------------------------------------------------------
Check if character is lowercase letter
-----------------------------------------------------------------*/
bool character::islower(const char s) {
  return (s >= 'a' && s <= 'z') ? true : false ;
}

/*-----------------------------------------------------------------
convert s to uppercase and return upper case equivalent
-----------------------------------------------------------------*/
char character::toupper(char s) {
  if (islower(s)) {
```

```
    s = (s - 'a') + 'A' ;
  }
  return s ;
}

/*----------------------------------------------------------------
convert s to lowercase and return lower case equivalent
----------------------------------------------------------------*/
char character::tolower(char s) {
  if (isupper(s)) {
    s = (s - 'A') + 'a' ;
  }
  return s ;
}

//EOF

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: charactertest.cpp

On linux:
g++ character.cpp charactertest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test character object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "character.h"

/*-----------------------------------------------------------------
2. ch d = 5 is NOT an alphabet
3. ch d = 5 is an alpha numeric character
4. ch P = P is a upper case character
5. ch u = u is  NOT a upper case character
6. ch u = u is a lower case character
8. ch u = U After converting to upper U
9. ch u = U After converting to upper U
P = P
10.ch P = p After converting to lower p
11.ch P = p After converting to lower p
ha ha it is ascii
-----------------------------------------------------------------*/
void test1() {
  char d = '5' ;
  char P = 'P' ;
  char u = 'u' ;

  if (!(character::isalpha(u))) {
    cout << "1. ch u is an alphabet\n" ;
  }
  if (!(character::isalpha(d))) {
    cout << "2. ch d = " << d << " is NOT an alphabet\n" ;
  }
  if (character::isalnum(d)) {
    cout << "3. ch d = " << d << " is an alpha numeric character\n" ;
  }

  if (character::isupper(P)) {
    cout << "4. ch P = " << P << " is a upper case character\n" ;
  }
  if (!(character::isupper(u))) {
    cout << "5. ch u = " << u << " is  NOT a upper case character\n" ;
  }

  if (character::islower(u)) {
    cout << "6. ch u = " << u << " is a lower case character\n" ;
  }
  if (!(character::islower(u))) {
    cout << "7. ch u = " << u << " is  NOT a lower case character\n" ;
  }

  cout << "8. ch u = " << u << " After converting to upper " << (u = character::toupper(u)) << endl ;
  cout << "9. ch u = " << u << " After converting to upper " << (u = character::toupper(u)) << endl ;
  cout << "P = " << P << endl ;
  cout << "10.ch P = " << P << " After converting to lower " << (P = character::tolower(P)) << endl ;
  cout << "11.ch P = " << P << " After converting to lower " << (P = character::tolower(P)) << endl ;
```

```
  if (character::is_ascii(u)) {
    cout << "ha ha it is ascii\n";
  }
}

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  test1() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF
```

VASUDEVAMURTHY

2.3 C Strings

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: carray.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has carray class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef carray_H
#define carray_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of carray class
-----------------------------------------------------------------*/
class carray{
public:
  //length
  static int strlen(const char* s) ;
  static int intlen(int i) ;

  //Conversion
  static int atoi(const char* s) ;
  static void itoa(int i, char* s) ;

  //compare
  static int strcmp(const char* s1, const char* s2) ;
  static int strncmp(const char* s1, const char* s2,int n) ;
  static int strcasecmp(const char* s1, const char* s2) ;
  static int strcasencmp(const char* s1, const char* s2,int n) ;

  //copy
  static void strcpy(char* dest, const char* src) ;
  static void strncpy(char* dest, const char* src,int n) ;

  //concatenation
  static void strcat(char* dest, const char* src) ;
  static void strncat(char* dest, const char* src,int n) ;
  //find
  static const char* strstr(const char* s, const char* find) ;
private:

};

```

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: carray.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has enumeration class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "carray.h"

/*-----------------------------------------------------------------
Definition of routines of carray class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
Give length of the string excluding NULL
01234
s = jag\0  -- U require 4 memory location to store
return 3 ;
length does *NOT* include '\0'
-----------------------------------------------------------------*/
int carray::strlen(const char* s) {
  int l = 0 ;
  while (*s) {
    l++ ;
    s++ ;
  }
  return l ;
}

/*-----------------------------------------------------------------
4 -> 1
-89 -> 3

-----------------------------------------------------------------*/
int carray::intlen(int i) {
  int l = 0 ;
  cout << "WRITE THE CODE\n" ;
  return l ;
}

/*-----------------------------------------------------------------
"73" --> 73
"-4"  --> -4
"7a" --> 0
```

```
"-" --> 0
When atoi encounters a string with non numerical sequence,
it returns zero (0).
----------------------------------------------------------------*/
int carray::atoi(const char* s) {
  int l = 0 ;
  bool minus = false ;
  if (s) {
    if (*s == '-') {
      minus = true ;
      s++ ;
    }else {
      //do not increment s
    }
  }
  if (s) { //the input can be only s = "-"
    do {
      if (*s >= '0' && *s <= '9') {
        l = l * 10 + (*s - '0') ;
        s++ ;
      }else {
        return 0 ;
      }
    }while (*s) ;
    return (minus) ? -l : l ;
  }
}

/*----------------------------------------------------------------
Converts
79 to "79\0"
8a to "\0"
-78 to "-78\0"
It is assumed that user gives s array which is >= length of i + 1
----------------------------------------------------------------*/
void carray::itoa(int i, char* s) {
  cout << "WRITE THE CODE\n" ;
}

/*----------------------------------------------------------------
Compares the two strings s1 and s2,
case sensitive: a and A are different.

0 : (s1 == s2)   abc = abc
- : (s1 < s2)    ABC < abc
+ : (s1 > s2)    abc > ABC
----------------------------------------------------------------*/
int carray::strcmp(const char* s1, const char* s2) {
  while (s1 && s2) {
```

```
    if (*s1 == *s2) {
      if (*s1 == '\0') { //that means *s1 = \0 && *s2 = \0
        return 0 ;
      }
      s1++ ;
      s2++ ;
    }else {
      return (*s1 - *s2) ;
    }
  }
  return 0 ;
}

/*-----------------------------------------------------------------
Compares first n character of s1 with s2
case sensitive: a and A are different.

0 : (s1 == s2)
- : (s1 < s2)
+ : (s1 > s2)
-----------------------------------------------------------------*/
int carray::strncmp(const char* s1, const char* s2, int n) {
  cout << "WRITE THE CODE\n" ;

  return 0 ;
}

/*-----------------------------------------------------------------
Compares the two strings s1 and s2,
ignoring the case of the characters.

0 : (s1 == s2)  abc = ABC
- : (s1 < s2)   ab < xb
+ : (s1 > s2)   xb > ab
-----------------------------------------------------------------*/
int carray::strcasecmp(const char* s1, const char* s2) {
  while (s1 && s2) {
    char a = *s1 ;
    a = tolower(a) ;
    char b = *s2 ;
    b = tolower(b) ;
    if (a == b) {
      if (a == '\0') { //that means *s1 = \0 && *s2 = \0
        return 0 ;
      }
      s1++ ;
      s2++ ;
    }else {
      return (a - b) ;
```

```
    }
  }
  return 0 ;
}

/*----------------------------------------------------------------
Compares first n character of s1 with s2
ignoring the case of the characters.

0 : (s1 == s2)
- : (s1 < s2)
+ : (s1 > s2)
----------------------------------------------------------------*/
int carray::strcasencmp(const char* s1, const char* s2, int n) {
  cout << "WRITE THE CODE\n" ;

  return 0 ;
}

/*----------------------------------------------------------------
src = "jag"
dest = "ucsc" ;

dest = "jag"

To avoid overflows, the size of the array pointed by destination
shall be long enough to contain the same C string as source
(including the terminating null character), and
should not overlap in memory with source.
----------------------------------------------------------------*/
void carray::strcpy(char* dest, const char* src) {
  int i = 0 ;
  char ch = 'a' ;
  do {
    ch = src[i] ;
    dest[i] = ch ;
    ++i ;
  }while (ch) ;
}

/*----------------------------------------------------------------
same as strcpy, but first n characters of src is copied to dest
----------------------------------------------------------------*/
void carray::strncpy(char* dest, const char* src, int n) {
  cout << "WRITE THE CODE\n" ;

}

/*----------------------------------------------------------------
```

```
/*----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: carraytest.cpp

On linux:
g++ carray.cpp carraytest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test carray object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "carray.h"

/*----------------------------------------------------------------

----------------------------------------------------------------*/
void test1() {
  cout << "1. length(\"jag\") = " << carray::strlen("jag") << endl ;
  cout << "2. length(\"jag\\) = " << carray::strlen("jag\\") << endl ;
  cout << "3. length(\"\") = " << carray::strlen("") << endl ;
  cout << "4. atoi(\"8602\") = " << carray::atoi("8602") << endl ;
  cout << "5. atoi(\"86z2\") = " << carray::atoi("86z2") << endl ;
  cout << "6. atoi(\"-8052\") = " << carray::atoi("-8052") << endl ;
  cout << "7. atoi(\"-\") = "  << carray::atoi("-") << endl ;
  cout << "8. atoi(\"-7\") = "  << carray::atoi("-7") << endl ;
  cout << "9. atoi(\"7-6\") = "  << carray::atoi("7-6") << endl ;
  cout << "10. strcmp(abc,abc) = "  << carray::strcmp("abc","abc") << endl ;
  cout << "11. strcmp(ABC,abc) = "  << carray::strcmp("ABC","abc") << endl ;
  cout << "12. strcmp(abc,ABC) = "  << carray::strcmp("abc","ABC") << endl ;
  cout << "13. strcmp("","") = "  << carray::strcmp("","") << endl ;
  cout << "14. strcmp("",ABC) = "  << carray::strcmp("","ABC") << endl ;
  cout << "15. strcmp(ABC,"") = "  << carray::strcmp("ABC","") << endl ;
  cout << "16. strcasecmp(abc,ABC) = "  << carray::strcasecmp("abc","ABC") << endl ;
  cout << "17. strcmp(ab,xb) = "  << carray::strcasecmp("ab","xb") << endl ;
  cout << "18. strcmp(xb,ab) = "  << carray::strcasecmp("xb","ab") << endl ;


  /* Write test program to test all other functions */
}

/*----------------------------------------------------------------
test bed
```

2.4 Problem set

Problem 2.4.1. Fill the code in the program **carray.cpp**. Write **test programs** in **carraytest.cpp** routine to test your code.

Problem 2.4.2. This is an assignment problem for my 3rd grade daughter at Tom Matsumoto Elementary School, San Jose, CA.

```
Problem: The cost of each letter is given in the table below.
Find a SINGLE LEGEAL WORD (that must be in a dictionary) such that the total
cost of the word is EXACTLY 25$.

Example:

begin 5.17
beginner 7.42
beginners 7.57

The cost of each alphabet is as follows:
a = 1.00$
b = 0.01$
c = 0.02$
d = 0.03$
e = 2.00$
f = 0.04$
g = 0.05$
h = 0.06$
i = 3.00$
j = 0.07$
k = 0.08$
l = 0.09$
m = 0.10$
n = 0.11$
o = 4.00$
p = 0.12$
q = 0.13$
r = 0.14$
s = 0.15$
t = 0.16$
u = 5.00$
v = 0.17$
```

VASUDEVAMURTHY

```
w = 0.18$
x = 0.19$
y = 6.00$
z = 0.20$
```

Write a *class* called **TomMatsumoto** that can print words that satisfies the above requirements.

VASUDEVAMURTHY

Chapter 3

Concept of reference

3.1 Introduction

VASUDEVAMURTHY

Figure 3.1: Alias and pointers

VASUDEVAMURTHY

3.2 Concept of swapping using reference

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Figure 3.2: Concept of call by value, address and reference

VASUDEVAMURTHY

3.3 FAQ on references

https://www.hasustorm.com/books/English/Addison.Wesley.Cpp.FAQs.2nd.Edition.internal.eBook-LiB.chm/0201309831_ch11lev1sec4.html

FAQ 11.04 What happens when a value is assigned to a reference?

The reference remains bound to the same referent,
and the value of the referent is changed.

```
void f(int& x, int* y, int z) {
  x = 5;   //main()'s i changed to 5
  *y = 6;  //main()'s i changed to 6
  z = 7;   //no change to main()'s i
}

int main()
{
  int i = 4;
  f(i, &i, i);
}
```

Figure 3.3: FAQ 1

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: reference.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has reference class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef reference_H
#define reference_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of reference class
-----------------------------------------------------------------*/
class reference{
public:
  void test_swap();
  void test_faq();
  void test_return_by_reference();

private:
  void _print(const int a, const int b, const char *s);
  void _call_by_value(int a, int b);
  void _call_by_address(int* a, int* b);
  void _call_by_reference(int& a, int& b);
  void _f(int& x, int* y, int z) ;
  void _faq1() ;
  void _faq2() ;
  void _faq4() ;
  int& _f2();


};


#endif
//EOF
```

VASUDEVAMURTHY

https://www.hasustorm.com/books/English/Addison.Wesley.Cpp.FAQs.2nd.Edition.internal.eBook-LiB.chm/0201309831_ch11lev1sec8.html

FAQ 11.08 Can a reference be made to refer to a different referent?

No, it can't.

Unlike a pointer, once a reference is bound to an object,
it cannot be made to refer to a different object.
The alias cannot be separated from the referent.

For example, the last line of the following example changes i to 6;
it does not make the reference k refer to j.
Throughout its short life, k will always refer to i.

```
int main()
{
  int  i = 5;
  int  j = 6;
  int& k = i;  <-- 1
       k = j;   <-- 2
}
```

(1) Bind k so it is an alias for i
(2) Change i to 6—does NOT bind k to j

Figure 3.4: FAQ 2

https://www.hasustorm.com/books/English/Addison.Wesley.Cpp.FAQs.2nd.Edition.internal.eBook-LiB.chm/0201309831_ch11lev1sec11.html

FAQ 11.11 When are pointers needed?

References are usually preferred over pointers when aliases
are needed for existing objects,
making them useful in parameter lists and as return values.

Pointers are required when it might be necessary to change the binding
to a different referent or to refer to a nonobject (a NULL pointer).
Pointers often show up as local variables and member objects.

Figure 3.5: FAQ 3

VASUDEVAMURTHY

https://www.hasustorm.com/books/English/Addison.Wesley.Cpp.FAQs.2nd.Edition
.internal.eBook-LiB.chm/0201309831_ch11lev1sec10.html

FAQ 11.10 Aren't references just pointers in disguise?

No, they are not.

It is important to realize that references and pointers are quite different.
A pointer should be thought of as a separate object with its own distinct set of operations
(*p, p->blah, and so on). So creating a pointer creates a new object.

In contrast, creating a reference does not create a new object;
it merely creates an alternative name for an existing object.
Furthermore the operations and semantics for the reference are defined by the referent;
references do not have operations of their own.

In the following example, notice that assigning 0 to the reference j is very different than
assigning 0 to the pointer p (the 0 pointer is the same as the NULL pointer).

```
int main()
{
  int  i = 5;
  int& j = i;      <-- 1
  int* p = &i;     <-- 2

  j = 67;    <-- 3
  p = 0;    <-- 4
}
```

(1) j is an alias for i
(2) p is a new object, not an alias
(3) Changes i. At this point both i and j are 67
(4) Changes p; does not affect i

Figure 3.6: FAQ 4

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: reference.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has reference class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "reference.h"

/*-----------------------------------------------------------------
Definition of routines of reference class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void reference::_print(const int a, const int b, const char *s) {
  cout << s << " a = " << a << " b = " << b << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void reference::_call_by_value(int a, int b) {
  int t = a ;
  a = b ;
  b = t ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void reference::_call_by_address(int* a, int* b) {
  int t = *a ;
  *a = *b ;
  *b = t ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void reference::_call_by_reference(int& a, int& b) {
  int t = a ;
  a = b ;
  b = t ;
}

/*-----------------------------------------------------------------
Before call_by_value a = 55 b = 69
after call_by_value a = 55 b = 69
Before _call_by_address a = 55 b = 69
after _call_by_address a = 69 b = 55
Before _call_by_reference a = 55 b = 69
after _call_by_reference a = 69 b = 55
-----------------------------------------------------------------*/
void reference::test_swap() {
  int a = 55;
  int b = 69 ;
  _print(a,b,"Before call_by_value") ;
  _call_by_value(a,b) ;
  _print(a,b,"after call_by_value") ;
```

VASUDEVAMURTHY

```cpp

  a = 55; b = 69 ;
  _print(a,b,"Before _call_by_address") ;
  _call_by_address(&a,&b) ;
  _print(a,b,"after _call_by_address") ;

  a = 55; b = 69 ;
  _print(a,b,"Before _call_by_reference") ;
  _call_by_reference(a,b) ;
  _print(a,b,"after _call_by_reference") ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void reference::_f(int& x, int* y, int z) {
  x = 5;
  *y = 6;
  z = 7;
}

/*-----------------------------------------------------------------
value of i before calling _f = 4
value of i  after calling _f = 6
-----------------------------------------------------------------*/
void reference::_faq1() {
  int i = 4;
  cout << "value of i before calling _f = " << i << endl ;
  _f(i, &i, i);
  cout << "value of i  after calling _f = " << i << endl ;
}

/*-----------------------------------------------------------------
Before k = j: i = 5 j = 6 k = 5
After k = j: i = 6 j = 6 k = 6
-----------------------------------------------------------------*/
void reference::_faq2() {
  int i = 5;
  int j = 6;
  int& k = i;
  cout << "Before k = j: " << "i = " << i << " j = " << j << " k = " << k << endl ;
  k = j;
  cout << "After k = j: " << "i = " << i << " j = " << j << " k = " << k << endl ;
}

/*-----------------------------------------------------------------
Before j = 67, p = NULL: i = 5 j = 5 p = 0014E9B8
After  j = 67, p = NULL: i = 67 j = 67 p = 00000000
-----------------------------------------------------------------*/
void reference::_faq4() {
  int i = 5;
  int& j = i;
  int* p = &i ;
  cout << "Before j = 67, p = NULL: " << "i = " << i << " j = " << j << " p = " << p << endl ;
  j = 67 ;
  p = NULL ;
  cout << "After  j = 67, p = NULL: " << "i = " << i << " j = " << j << " p = " << p << endl ;
}

/*-----------------------------------------------------------------
Can you return by reference
-----------------------------------------------------------------*/
int& reference::_f2() {
  int z = 897868;
  cout << "z = " << z << endl ;
  return z ;
```

```
}

/*-----------------------------------------------------------------
z = 897868

==21355== Use of uninitialised value of size 8
==21355==    at 0x3251C793FB: ??? (in /usr/lib64/libstdc++.so.6.0.8)
==21355==    by 0x3251C895D4: std::ostreambuf_iterator<char, std::char_traits<ch
May not work
-----------------------------------------------------------------*/
void reference::test_return_by_reference() {
  int& a = _f2() ;
  cout << "a = " << a << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
void reference::test_faq(){
  _faq1() ;
  _faq2() ;
  _faq4() ;

}

//EOF

```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: referencetest.cpp

On linux:
g++ reference.cpp referencetest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test reference object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "reference.h"

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  reference a ;
  a.test_swap() ;
  a.test_faq() ;
  a.test_return_by_reference();
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

3.4 Passing char* by value, address and reference

```
//filename: C:\work\software\objects\hanging\charstarref.cpp
#include "../util/util.h"

const char* a[] = {"MIT", "UCB"};

void f(const char* d) {
  d = a[1];
  cout << "d = " << d << endl;
}

void f(const char** d) {
  *d = a[1];
  cout << "d = " << *d << endl;
}

void f1(const char*& d) {
  d = a[1];
  cout << "d = " << d << endl;
}

int main() {
  const char* c = a[0];
  cout << "c1 = " << c << endl;
  f(c);
  cout << "c2 = " << c << endl;
  cout << "-------------------\n";
  cout << "c3 = " << c << endl;
  f(&c);
  cout << "c4 = " << c << endl;
  cout << "-------------------\n";
  c = a[0];
  cout << "c5 = " << c << endl;
  f1(c);
  cout << "c6 = " << c << endl;
  cout << "-------------------\n";
  return 0;
}
```

Figure 3.7: passing char* by value, address and reference

VASUDEVAMURTHY

VASUDEVAMURTHY

Chapter 4

Dynamic memory allocation

4.1 Introduction

4.2 Static and dynamic memory allocation

VASUDEVAMURTHY

Figure 4.1: Static and dynamic memory allocation

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: dynamic.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has dynamic class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef dynamic_H
#define dynamic_H

#include "../util/util.h"
#include "../carray/carray.h" //for strcpy

/*-----------------------------------------------------------------
Declaration of dynamic class
-----------------------------------------------------------------*/
class dynamic {
public:
  void test_all();
private:
  void _test_int();
  void _test_char();
  void _test_int_array(int size);
  void _test_char_array(int size);
  void _test_obj(int size);
};


#endif
//EOF
```

```cpp
/*----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: dynamic.cpp
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has dynamic class definition
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "dynamic.h"

/*----------------------------------------------------------------
Definition of routines of dynamic class
----------------------------------------------------------------*/

/*----------------------------------------------------------------
address of p = 007C7160 and the content of p is = 27
----------------------------------------------------------------*/
void dynamic::_test_int() {
  int* p = new int(27);
  cout << "address of p = " << p << " and the content of p is = " << *p << endl;
  delete p;
}

/*----------------------------------------------------------------
content of p is = H
----------------------------------------------------------------*/
void dynamic::_test_char() {
  char* p = new char('H');
  cout << "content of p is = " << *p << endl;
  delete p;
}

/*----------------------------------------------------------------
p[0] = 1 p[1] = 2 p[2] = 3 p[3] = 4 p[4] = 5
p = abcd
----------------------------------------------------------------*/
void dynamic::_test_int_array(int size) {
  int* p = new int[size];
  for (int i = 0; i < size; i++) {
    p[i] = i + 1;
  }
  for (int i = 0; i < size; i++) {
    cout << " p[" << i << "] = " << p[i];
  }
  cout << endl;
  delete[] p;
}
```

```

/*-----------------------------------------------------------------
p = abcd
-----------------------------------------------------------------*/
void dynamic::_test_char_array(int size) {
  char* p = new char[size];
  strcpy(p, "abcd");
  cout << "p = " << p << endl;
  delete[] p;
}

/*-----------------------------------------------------------------
Student id = Z567 Student age = 25
-----------------------------------------------------------------*/
void dynamic::_test_obj(int size) {
  class student {
  public:
    int age;
    char *id;
  };
  student* a = new student;
  a->age = 25;
  a->id = new char[size];
  strcpy(a->id, "Z567");
  cout << "Student id = " << a->id << " Student age = " << a->age << endl;
  delete[] a->id;
  delete a;
}

/*-----------------------------------------------------------------
test all
-----------------------------------------------------------------*/
void dynamic::test_all() {
  _test_int();
  _test_char();
  _test_int_array(5);
  _test_char_array(5);
  _test_obj(5);
}


//EOF

```

```cpp
/*----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: dynamictest.cpp

On linux:
g++ ../carray/carray.cpp dynamic.cpp dynamictest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test dynamic object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "dynamic.h"

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  dynamic a;
  a.test_all();
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testbed();
  return 0;
}

//EOF

```

4.3 Allocating two dimensional array

4.3.1 Static allocation

Figure 4.2: Static allocation of two dimensional array

4.3.2 Dynamic allocation

VASUDEVAMURTHY

DYNAMIC TWO DIMENSIONAL ARRAY (METHOD 1)

```
int Row = 3 ; //Can be variable
const int Col = 2 ; //MUST BE CONSTANT

int (*x)[Col] = new int[Row][Col] ;
```

Dynamic

Must know at compilation

x is a int pointer to an array of size(Row * Col) ;

x
000000FF1E8FF888

	Col 0	Col 1
Row 0	x[0,0] 0	x[0,1] 1
Row 1	x[1,0] 10	x[1,1] 11
Row 2	x[2,0] 20	x[2,1] 21

Because x is a single dimensional array you can delete using:

```
delete [] x ;
```

x[r][c] = *(*(x+r)+c)

000000FF1E8FF888	x[0,0]	0
000000FF1E8FF88C	x[0,1]	1
000000FF1E8FF890	x[1,0]	10
000000FF1E8FF894	x[1,1]	11
000000FF1E8FF898	x[2,0]	20
000000FF1E8FF89C	x[2,1]	21

Internally, a two dimensional array is just a collection of 1-dimensional array

Column must be known to the compiler. CANNOT BE VARAIBLE

Because, Internally, a two dimensional array is just a collection of 1-dimensional array

```
static void print_array(const int a[][2] , int R, int C) {
  for (int r = 0 ; r < R; r++) {
    for (int c = 0; c < C; c++) {
      cout << a[r][c] << " " ;
    }
    cout << endl ;
  }
}
```

Accessing individual elements

You cannot write a program that takes number of rows and number of columns from user and allocate a dynamic array using this form:

```
int *(x)[Col] = new int [row][Col]
```

Figure 4.3: Dynamic allocation of two dimensional array - Method 1

VASUDEVAMURTHY

DYNAMIC TWO DIMENSIONAL ARRAY (METHOD 2)

```
int R = 3 ;
int C = 2 ;
```

Both Row and col are dynamic

```
int **x = new int*[R] ;
for (int i = 0; i < R; ++i) {
    x[i] = new int[C] ;
}
```

Allocation

```
//Here x is a pointer to an array of R pointers.
//Each pointer is pointing to a dynamic array of size C
//Note that the data is garbage
```

```
for (int i = 0; i < R; ++i) {
  delete [] x[i] ;
}
delete [] x ;
```

deletion

```
void two2darray::print_array(const int** a , int R, int C) {
  for (int r = 0 ; r < R; r++) {
    for (int c = 0; c < C; c++) {
      cout <<  a[r][c] << " " ;
    }
    cout << endl ;
  }
}
```

Must pass R and C

Access

Figure 4.4: Dynamic allocation of two dimensional array - Method 2

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: two2darray.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has two2darray class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef two2darray_H
#define two2darray_H

#include "../util/util.h"

/*-----------------------------------------------------------------
Declaration of two2darray class
-----------------------------------------------------------------*/
class two2darray {
public:
  void test_static_2d();
  void test_dynamic_2d(int R);
  void test_dynamic_2d(int R, int C);
private:
  void _print_array(const char*s, const int a[][2], int R, int C);
  void _print_array(const char*s, int** a, int R, int C);
  void _add_contents_by(int a[][2], int n, int R, int C);
  void _add_contents_by(int** a, int n, int R, int C);
  void _fill(int a[][2]);
  void _fill(int**a, int R, int C);
};


#endif

//EOF

```

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: two2darray.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has two2darray class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "two2darray.h"

/*-----------------------------------------------------------------
Definition of routines of two2darray class
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
print two dimensional array
const int C = 2 ; //must be constant and known at the time of compilation
//declare x as pointer to array of int
int (*x)[C]  = new int[R][C];
-----------------------------------------------------------------*/
void two2darray::_print_array(const char* s, const int a[][2], int R, int C
) {
  cout << s << endl;
  cout << "size of a = " << sizeof(a) << endl;
  for (int r = 0; r < R; r++) {
    cout << "Row " << r << "    " << &(a[r]) << "   ";
    for (int c = 0; c < C; c++) {
      cout << &(a[r][c]) << "     " << a[r][c] << " ";
    }
    cout << endl;
  }
  cout << "To show: a[r][c] = *(*(a+r)+c) " << endl;
  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      cout << &(*(*(a + r) + c)) << " = " << *(*(a + r) + c) << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
print two dimensional array
int **x = new int*[R] ;
for (int i = 0; i < C; ++i) {
  x[i] = new int[C] ;
}
-----------------------------------------------------------------*/
void two2darray::_print_array(const char*s, int** a, int R, int C) {
```

```
  cout << s << endl;
  cout << "size of a = " << sizeof(a) << endl;
  for (int r = 0; r < R; r++) {
    cout << "Row " << r << "    " << &(a[r]) << "   ";
    for (int c = 0; c < C; c++) {
      cout << &(a[r][c]) << "     " << a[r][c] << " ";
    }
    cout << endl;
  }
  cout << "To show: a[r][c] = *(*(a+r)+c) " << endl;
  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      cout << &(*(*(a + r) + c)) << " = " << *(*(a + r) + c) << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
add contents by n

const int C = 2 ; //must be constant and known at the time of compilation
//declare x as pointer to array of int
int (*x)[C]  = new int[R][C];
-----------------------------------------------------------------*/
void two2darray::_add_contents_by(int a[][2], int n, int R, int C) {
  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      a[r][c] += n;
    }
  }
}

/*-----------------------------------------------------------------
add contents by n
int **x = new int*[R] ;
for (int i = 0; i < C; ++i) {
  x[i] = new int[C] ;
}
-----------------------------------------------------------------*/
void two2darray::_add_contents_by(int** a, int n, int R, int C) {
  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      a[r][c] += n;
    }
  }
}

/*-----------------------------------------------------------------
fill the dynamic array by predefined array
const int C = 2 ; //must be constant and known at the time of compilation
//declare x as pointer to array of int
```

```
int (*x)[C]  = new int[R][C];
-----------------------------------------------------------------*/
void two2darray::_fill(int a[][2]) {
  const int R = 3;
  const int C = 2;
  int x[R][C] = { {0,1},{10,11},{20,21} };

  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      a[r][c] = x[r][c];
    }
  }
}

/*-----------------------------------------------------------------
fill the dynamic array by predefined array
int **x = new int*[R] ;
for (int i = 0; i < C; ++i) {
  x[i] = new int[C] ;
}
-----------------------------------------------------------------*/
void two2darray::_fill(int**a, int R, int C) {
  int x[3][2] = { {0,1},{10,11},{20,21} };

  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      a[r][c] = x[r][c];
    }
  }
}

/*-----------------------------------------------------------------
Test static two dimensional array
c0 c1
row0: 00 01
row1: 10 11
row2: 20 21
-----------------------------------------------------------------*/
void two2darray::test_static_2d() {
  const int R = 3;
  const int C = 2;
  int x[R][C] = { {0,1},{10,11},{20,21} };
  cout << "Size of x = " << sizeof(x) << endl;
  int num = sizeof(x) / sizeof(int);
  cout << "number of elements = " << num << endl;
  //int* p = x ; //This will not compile
  _print_array("1. before", x, R, C);
  _add_contents_by(x, 5, R, C);
  _print_array("2.After adding 5", x, R, C);
}
```

```
/*-----------------------------------------------------------------
Creating dynamic array, R is dynamic C is static
      c0 c1
row0: 00 01
row1: 10 11
row2: 20 21

-----------------------------------------------------------------*/
void two2darray::test_dynamic_2d(int R) {
  const int C = 2; //must be constant and known at the time of compilation
  int(*x)[C] = new int[R][C];
  //Here x is a pointer to an array of (R*C) integers
  cout << "Size of x = Size of ptr = " << sizeof(x) << endl;
  _fill(x);
  _print_array("1. before", x, R, C);
  _add_contents_by(x, 5, R, C);
  _print_array("2.After adding 5", x, R, C);
  delete[] x;
}

/*-----------------------------------------------------------------
Creating dynamic array, R and C are dynamic
      c0 c1
row0: 00 01
row1: 10 11
row2: 20 21
-----------------------------------------------------------------*/
void two2darray::test_dynamic_2d(int R, int C) {
  int **x = new int*[R];
  cout << "NUM ALLOCATION = " << _msize(x) << " Bytes" << endl;
  for (int i = 0; i < R; ++i) {
    x[i] = new int[C];
    cout << "NUM ALLOCATION = " << _msize(x[i]) << " Bytes" << endl;
  }
  _fill(x, R, C);
  _print_array("1. before", x, R, C);
  _add_contents_by(x, 5, R, C);
  _print_array("2.After adding 5", x, R, C);
  for (int i = 0; i < R; ++i) {
    delete[] x[i];
  }
  delete[] x;
}


//EOF
```

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: intarraytest.cpp

On linux:
g++ intarray.cpp intarraytest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test intarray object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "intarray.h"


/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  intarray a;
  a.array_basic1();
  a.array_basic2();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed();
  return 0;
}

//EOF
```

4.4 *unique_ptr*

Allocating 1D and 2D array using unique_ptr

```
{
  int* x = new int[N] ;

  delete [] x ;
}
```

```
{
  unique_ptr<int[]> p = make_unique<int[]>(N) ;
}
```

```
void two2darray::test_dynamic_2d(int R, int C) {
  int **x = new int*[R];
  for (int i = 0; i < R; ++i) {
    x[i] = new int[C];
  }
  _fill(x, R, C);
  _print_array("After fill", x, R, C);
  for (int i = 0; i < R; ++i) {
    delete x[i];
  }
  delete[] x;
}
```

```
void uniqueptr::twoDimensionArray(int R, int C) {
  unique_ptr<unique_ptr<int[]>[]> x =
      make_unique< unique_ptr<int[]>[] >(R);
  for (int i = 0; i < R; ++i) {
    x[i] = make_unique<int[]>(C);
  }
  fill(x, R, C);
  print_array("After Fill", x, R, C);
}
```

Figure 4.5: Dynamic allocation of arrays using *unique_ptr*

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: uniqueptr.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has uniqueptr class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "uniqueptr.h"

/*-----------------------------------------------------------------
basic tests
-----------------------------------------------------------------*/
void uniqueptr::basic() {
  {
    //TO SHOW unique_ptr has NO copy constructor and equal operator

    unique_ptr <complex> p1 = make_unique<complex>(5, 6);
    cout << "p1 = " << p1.get() << endl;
    unique_ptr <complex> p3{ make_unique<complex>(1,-3) };
    cout << "p3 = " << p3.get() << endl;
    auto p4{ make_unique<complex>(-2,-4) };
    cout << "p4 = " << p4.get() << endl;

    //both copy constructors and delete opera
    //unique_ptr <complex> p2 = p1 ;  COPY
    //unique_ptr <complex> p2(p1) ;  COPY
    //p3 = p1 ; //EQUAL
  }
  if (0)
  {
    //TO SHOW MEMORY LEAK

    unique_ptr <complex> p1{ make_unique<complex>(5,6) };
    cout << "p1 = " << p1.get() << endl;
    p1.release(); //Looses complex pointer
    assert(p1.get() == nullptr);
  }

  {
    //TO SHOW HOW TO GIVE RESPONBILTY

    unique_ptr <complex> p1{ make_unique<complex>(5,6) };
    cout << "p1 = " << p1.get() << endl;
    p1.reset();
    assert(p1.get() == nullptr);
    p1 = make_unique<complex>(1, -3);
    cout << "p1 = " << p1.get() << endl;
```

Has no copy and equal operator

```
in complex constructor: x = 5 y = 6
p1 = 5+i6
in complex constructor: x = 1 y = -3
p3 = 1-i3
in complex constructor: x = -2 y = -4
p4 = -2-i4
In complex Destructor -2-i4
In complex Destructor 1-i3
In complex Destructor 5+i6
```

release will clear complex pointer. Will not call destructor

reset will call complex destructor

```
in complex constructor: x = 5 y = 6
p1 = 5+i6
In complex Destructor 5+i6
in complex constructor: x = 1 y = -3
p1 = 1-i3
In complex Destructor 1-i3
```

```
53      }
54
55      {
56        //TO SHOW HOW TO GIVE RESPONBILTY through std::move
57
58        unique_ptr <complex> p1{ make_unique<complex>(5,6) };
59        cout << "p1 = " << p1.get() << endl;
60        p1 = (move(make_unique<complex>(1, -3)))
61        cout << "p1 = " << p1.get() << endl;
62      }
63
64      {
65        //TO SHOW HOW TO USE ARRAY
66        const auto N = 5;
67        auto p = make_unique<complex[]>(N);
68        for (auto i = 0; i < N; ++i) {
69          //What happens if you do: auto o = p[i] ;
70          auto& o = p[i]; //p[i] is overloaded in unique_ptr to give object
71          o.setxy(i, -i);
72        }
73      }
74    }
75
76    /*-----------------------------------------------------------
77    pass by value test
78    -----------------------------------------------------------
79    void uniqueptr::_f(unique_ptr <complex> p) {
80      cout << "p = " << p.get() << endl;
81    }
82
83    /*-----------------------------------------------------------------
84    pass by reference test
85    -----------------------------------------------------------------*/
86    void uniqueptr::_f1(unique_ptr <complex>& p) {
87      cout << "p = " << p.get() << endl;
88    }
89
90    /*-----------------------------------------------------------------
91    pass by value test
92    -----------------------------------------------------------------*/
93    void uniqueptr::passByValue() {
94      unique_ptr <complex> p{ make_unique<complex>(5,6) };
95      cout << "p = " << p.get() << endl;
96      //_f(p) ; Will not compile as there is no copy constructor
97      _f(move(p));
98    }
99
100   /*-----------------------------------------------------------------
101   pass by Reference test
102   -----------------------------------------------------------------*/
103   void uniqueptr::passByReference() {
104     unique_ptr <complex> p{ make_unique<complex>(5,6) };
```

```
in complex constructor: x = 5 y = 6
p1 = 5+i6
in complex constructor: x = 1 y = -3
In complex Destructor 5+i6
p1 = 1-i3
In complex Destructor 1-i3
```

```
In complex constructor: x = 0 y = 0
in complex constructor: x = 0 y = 0
in complex constructor: x = 0 y = 0
in complex constructor: x = 0 y = 0
in complex constructor: x = 0 y = 0
In complex Destructor 4-i4
In complex Destructor 3-i3
In complex Destructor 2-i2
In complex Destructor 1-i1
In complex Destructor 0+i0
```

```
  cout << "p = " << p.get() << endl;
  _f1(p);
}

/*-----------------------------------------------------------------
return by value test
-----------------------------------------------------------------*/
unique_ptr <complex> uniqueptr::_f() {
  unique_ptr <complex> p{ make_unique<complex>(15,16) };
  cout << "p = " << p.get() << endl;
  return p;
}
/*-----------------------------------------------------------------
return by value test
-----------------------------------------------------------------*/
void uniqueptr::returnByValue() {
  auto p = _f();
  cout << "p = " << p.get() << endl;
}

//EOF

```

VASUDEVA

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: uniqueptr.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has uniqueptr class declaration
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef uniqueptr_H
#define uniqueptr_H

#include "../util/util.h"
#include "../complex/complex.h"
#include "memory" //For unique_ptr

/*-----------------------------------------------------------------
Declaration of uniqueptr class
-----------------------------------------------------------------*/
class uniqueptr {
public:
  void basic();
  void passByValue();
  void passByReference();
  void returnByValue();
  void singleDimensionArray(int n);
  void twoDimensionArray(int r, int c);
private:
  void _f(unique_ptr <complex> p);
  void _f1(unique_ptr <complex>& p);
  unique_ptr <complex> _f();
};

#endif
```

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: uniqueptr.cpp
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file has uniqueptr class definition
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "uniqueptr.h"

/*-----------------------------------------------------------------
basic tests
-----------------------------------------------------------------*/
void uniqueptr::basic() {
  {
    //TO SHOW unique_ptr has NO copy constructor and equal operator

    unique_ptr <complex> p1 = make_unique<complex>(5, 6);
    cout << "p1 = " << p1.get() << endl;
    unique_ptr <complex> p3{ make_unique<complex>(1,-3) };
    cout << "p3 = " << p3.get() << endl;
    auto p4{ make_unique<complex>(-2,-4) };
    cout << "p4 = " << p4.get() << endl;

    //both copy constructors and delete operator is deleted
    //unique_ptr <complex> p2 = p1 ;  COPY
    //unique_ptr <complex> p2(p1) ;  COPY
    //p3 = p1 ; //EQUAL
  }
  if (0)
  {
    //TO SHOW MEMORY LEAK

    unique_ptr <complex> p1{ make_unique<complex>(5,6) };
    cout << "p1 = " << p1.get() << endl;
    p1.release(); //Looses complex pointer
    assert(p1.get() == nullptr);
  }

  {
    //TO SHOW HOW TO GIVE RESPONBILTY

    unique_ptr <complex> p1{ make_unique<complex>(5,6) };
    cout << "p1 = " << p1.get() << endl;
    p1.reset();
    assert(p1.get() == nullptr);
    p1 = make_unique<complex>(1, -3);
    cout << "p1 = " << p1.get() << endl;
```

```
  }

  {
    //TO SHOW HOW TO GIVE RESPONBILTY through std::move

    unique_ptr <complex> p1{ make_unique<complex>(5,6) };
    cout << "p1 = " << p1.get() << endl;
    p1 = (move(make_unique<complex>(1, -3)));
    cout << "p1 = " << p1.get() << endl;
  }
}

/*-----------------------------------------------------------------
pass by value test
-----------------------------------------------------------------*/
void uniqueptr::_f(unique_ptr <complex> p) {
  cout << "p = " << p.get() << endl;
}

/*-----------------------------------------------------------------
pass by reference test
-----------------------------------------------------------------*/
void uniqueptr::_f1(unique_ptr <complex>& p) {
  cout << "p = " << p.get() << endl;
}

/*-----------------------------------------------------------------
pass by value test
-----------------------------------------------------------------*/
void uniqueptr::passByValue() {
  unique_ptr <complex> p{ make_unique<complex>(5,6) };
  cout << "p = " << p.get() << endl;
  //_f(p) ; Will not compile as there is no copy constructor
  _f(move(p));
}

/*-----------------------------------------------------------------
pass by Reference test
-----------------------------------------------------------------*/
void uniqueptr::passByReference() {
  unique_ptr <complex> p{ make_unique<complex>(5,6) };
  cout << "p = " << p.get() << endl;
  _f1(p);
}

/*-----------------------------------------------------------------
return by value test
-----------------------------------------------------------------*/
unique_ptr <complex> uniqueptr::_f() {
  unique_ptr <complex> p{ make_unique<complex>(15,16) };
  cout << "p = " << p.get() << endl;
  return p;
```

```
}
/*-----------------------------------------------------------------
return by value test
-----------------------------------------------------------------*/
void uniqueptr::returnByValue() {
  auto p = _f();
  cout << "p = " << p.get() << endl;
}

/*-----------------------------------------------------------------
1d array
-----------------------------------------------------------------*/
void uniqueptr::singleDimensionArray(int N) {
  {
    unique_ptr<int[]> p = make_unique<int[]>(N);
    //auto p = make_unique<int[]>(N);
    for (int i = 0; i < N; ++i) {
      //What happens if you do: auto o = p[i] ;
      auto& o = p[i]; //p[i] is overloaded in unique_ptr to give object
      cout << "a[i] = " << p[i] << " ";
    }
    cout << endl;
  }
  {
    auto p = make_unique<complex[]>(N);
    for (int i = 0; i < N; ++i) {
      //What happens if you do: auto o = p[i] ;
      auto& o = p[i]; //p[i] is overloaded in unique_ptr to give object
      o.setxy(i, -i);
      cout << "a[i] = " << o << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
fill the dynamic array by predefined array
-----------------------------------------------------------------*/
static void fill(unique_ptr<unique_ptr<int[]>[]>& a, int R, int C) {
  int x[3][2] = { {0,1},{10,11},{20,21} };

  for (int r = 0; r < R; r++) {
    for (int c = 0; c < C; c++) {
      a[r][c] = x[r][c];
    }
  }
}

/*-----------------------------------------------------------------
print two dimensional array
-----------------------------------------------------------------*/
static void print_array(const char*s, unique_ptr<unique_ptr<int[]>[]>& a,
```

```
int R, int C) {
  cout << s << endl;
  cout << "size of a = " << sizeof(a) << endl;
  for (int r = 0; r < R; r++) {
    cout << "Row " << r << "    " << &(a[r]) << "   ";
    for (int c = 0; c < C; c++) {
      cout << &(a[r][c]) << "     " << a[r][c] << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
2d array
Creating dynamic array, R and C are dynamic
c0 c1
row0: 00 01
row1: 10 11
row2: 20 21
-----------------------------------------------------------------*/
void uniqueptr::twoDimensionArray(int R, int C) {
  //int **x = new int*[R];
  unique_ptr<unique_ptr<int[]>[]> x = make_unique< unique_ptr<int[]>[] >(R);
  cout << "Size of x = " << sizeof(x) << " Bytes" << endl;
  for (int i = 0; i < R; ++i) {
    //x[i] = new int[C];
    x[i] = make_unique<int[]>(C);
  }
  fill(x, R, C);
  print_array("After Fill", x, R, C);
  /*
    for (int i = 0; i < R; ++i) {
      delete x[i];
    }
    delete[] x;
  */
}

//EOF
```

```cpp
/*----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: complex.cpp uniqueptrtest.cpp

On linux:
g++ uniqueptr.cpp uniqueptrtest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test uniqueptr object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "uniqueptr.h"

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  complex::set_display(true);
  uniqueptr a;
  a.basic();
  a.passByValue();
  a.passByReference();
  a.returnByValue();
  a.singleDimensionArray(5);
  a.twoDimensionArray(3, 2);
}


/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testbed();
  return 0;
}

//EOF
```

4.5 Problem set

Problem 4.5.1. Write a *class* called **sieve** for **generating N prime numbers** as described in Fig 4.6. You must write two files **sieve.h** and **sieve.cpp** and test your class using the provided file **sievetest.cpp** Your program must work for all the cases provided in **sievetest.cpp**. You cannot change anything in the file **sievetest.cpp**. **email** only **sieve.h** and **sieve.cpp**. Also include screen shot of the program output as a pdf file.

Figure 4.6: Finding all prime numbers

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
file: sievetest.cpp

On linux:
g++ sieve.cpp sievetest.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test sieve object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "sieve.h"

/*----------------------------------------------------------------
Compute prime up to n (all numbers that cannot divide n except 1 and n)

if n = 30
prime numbers are: 2 3 5 7  11 13 17 19 23 29
----------------------------------------------------------------*/
void prime(int n) {
  sieve s;
  int as = 0;
  int* a = s.school_method(n, &as);
  s.print("School Method", as, a);
  int ss = 0;
  int* b = s.sieve_of_eratosthenes(n, &ss);
  s.print("Sieve  Method", ss, b);
  assert(s.verify(as, a, ss, b));
  delete [] a;
  delete [] b;
}

/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  prime(30);
  prime(1000);
  prime(10000);
}

/*----------------------------------------------------------------
main
----------------------------------------------------------------*/
int main() {
  testbed();
  return 0;
}

//EOF
```

VASUDEVAMURTHY

Problem 4.5.2.

VASUDEVAMURTHY

COXETER MAGIC SQUARE

Start with 1 in the middle of the top row; then go up and left, assigning numbers in increasing order to empty squares; if you fall off the square imagine the same square as tiling the plane and continue; if a square is occupied, move down instead and continue.

Works for only odd number

6	1	8
7	5	3
2	9	4

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

Implement class magicsquare
and test for various odd values of n

1. Make sure the sum is same in all directions

e-mail magicsquare.h
magicsqaure.cpp
magicsquaretest.cpp
must use only ../util/util.h

Figure 4.7: Odd Magic square

VASUDEVAMURTHY

Problem 4.5.3.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Initilize the chess board as shown in the figure.
Write a display routine that prints the board configuration
SHOW SOLUTION AS BOTH STATIC AND DYNAMIC ALLOCATION

K	white king	k	black king
Q	white queen	q	black queen
R	white rook	r	black rook
B	white bishop	b	black bishop
N	white knight	n	black knight
P	white pawn	p	black pawn
—	empty square		

r	n	b	q	k	b	n	r
p	p	p	p	p	p	p	p
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
P	P	P	P	P	P	P	P
R	N	B	Q	K	B	N	R

Figure 4.8: A chess board

VASUDEVAMURTHY

Problem 4.5.4.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens attack each other.
Thus, a solution requires that no two queens share the same row, column, or diagonal

For a 1 x 1 board, there is one trivial solution:

N=1

For 2 x 2 and 3 x 3 boards, there are no solutions.

Output solution

```
----1---
------1-
-1------
-----1--
--1-----
1-------
---1----
-------1
```

The eight queens puzzle has 92 distinct solutions.
If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 unique (or fundamental) solutions.

Figure 4.9: N Queens Puzzle

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

VASUDEVAMURTHY

Chapter 5

Function overloading and default function arguments

5.1 Introduction

5.2 Function overloading

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: overload.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

void f(int i) {
  cout << "In void f(int i) " << i << endl ;
}

void f(bool i) {
  cout << "In void f(bool i) " << i << endl ;
}

void f(long i) {
  cout << "In void f(long i) " << i << endl ;
}

void f(float i) {
  cout << "In void f(float i) " << i << endl ;
}

void f(double i) {
  cout << "In void f(double i) " << i << endl ;
}
void f(char i) {
  cout << "In void f(char i) " << i << endl ;
}

int main() {
  f(10) ;
  int i = 2147483647 ;
  f(i) ;
  long l = 2147483647 ;
  f(l) ;
  f(true) ;
  f(false) ;
  double d = 1.7e308 ;
  f(d) ;
  float f1 = float(5.67) ;
  f(f1) ;
  f('a') ;
  return 0 ;
}

/*

In void f(int i) 10
In void f(int i) 2147483647
In void f(long i) 2147483647
In void f(bool i) 1
In void f(bool i) 0
In void f(double i) 1.7e+308
```

VASUDEVAMURTHY

5.3 Default function arguments

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: defaultfunc.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

void f1(const char *n, int m) {
  cout << "f1: Name " << n << " Marks " << m << endl ;
}

void f2(const char *n, int m, int b) {
  cout << "f2: Name " << n << " Marks " << m + b  << endl ;
}

void f3(const char *n, int m, int b, char g) {
  cout << "f3: Name " << n << " Marks " << m + b << " Grade " << g << endl ;
}

void f(const char *n, int m, int b = 0 , char g = ' ') {
  cout << "f: Name " << n  << " Marks " << m + b ;
  if (g != ' ') {
    cout << " Grade " << g ;
  }
  cout << endl ;
}

int main() {
  f1("John Smith",78) ;
  f2("John Smith",78,10) ;
  f3("John Smith",78,10,'B') ;
  f("John Smith",78) ;
  f("John Smith",78,10) ;
  f("John Smith",78,10,'B') ;
  return 0 ;
}

/*
f1: Name John Smith Marks 78
f2: Name John Smith Marks 88
f3: Name John Smith Marks 88 Grade B
f: Name John Smith Marks 78
f: Name John Smith Marks 88
f: Name John Smith Marks 88 Grade B
*/
```

VASUDEVAMURTHY

Chapter 6

C++ class

6.1 Introduction

6.2 Need for *class*

```
13 void code_without_classes() {
14     int a_age ;
15     char a_name[10] ;
16     bool a_is_male ;
17
18     int b_age ;
19     char b_name[10] ;
20     bool b_is_male ;
21
22     a_age = 17 ;
23     strcpy(a_name,"john") ;
24     a_is_male = true ;
25
26     b_age = 21 ;
27     strcpy(b_name,"debbie") ;
28     b_is_male = false ;
29
30     cout << a_name << ";Age = " << a_age << ";Sex = " << ((a_is_male)? "male" : "female") << endl ;
31     cout << b_name << ";Age = " << b_age << ";Sex = " << ((b_is_male)? "male" : "female") << endl ;
32 }
```

Figure 6.1: Need for a class

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c1.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
john;Age = 17;Sex = male
debbie;Age = 21;Sex = female
-----------------------------------------------------------------*/
void code_without_classes() {
   int a_age ;
   char a_name[10] ;
   bool a_is_male ;

   int b_age ;
   char b_name[10] ;
   bool b_is_male ;

   a_age = 17 ;
   strcpy(a_name,"john") ;
   a_is_male = true ;

   b_age = 21 ;
   strcpy(b_name,"debbie") ;
   b_is_male = false ;

   cout << a_name << ";Age = " << a_age << ";Sex = " << ((a_is_male)? "male" : "female") << endl ;
   cout << b_name << ";Age = " << b_age << ";Sex = " << ((b_is_male)? "male" : "female") << endl ;
}

/*-----------------------------------------------------------------
john;Age = 17;Sex = male
debbie;Age = 21;Sex = female
-----------------------------------------------------------------*/
void code_with_classes() {
   class group {
   public:
      int age ;
      char name[10] ;
      bool is_male ;
      void print() {
         cout <<  name << ";Age = " << age << ";Sex = " << ((is_male)? "male" : "female") << endl ;
      }
   };

   group a ;
   a.age = 17 ;
   strcpy(a.name,"john") ;
   a.is_male = true ;
   a.print() ;

   group b ;
```


VASUDEVAMURTHY

```
56     b.age = 21 ;
57     strcpy(b.name,"debbie") ;
58     b.is_male = false ;
59     b.print() ;
60 }
61
62 /*-----------------------------------------------------------------
63
64 -----------------------------------------------------------------*/
65 int  main() {
66     code_without_classes();
67     code_with_classes();
68     return 0 ;
69 }
```

VASUDEVAMURTHY

```
38 void code_with_classes() {
39     class group {
40     public:
41         int age ;
42         char name[10] ;
43         bool is_male ;
44         void print() {
45             cout << name << ";Age = " << age << ";Sex = " << ((is_male)? "male" : "female") << endl ;
46         }
47     };
48
49     group a ;
50     a.age = 17 ;
51     strcpy(a.name,"john") ;
52     a.is_male = true ;
53     a.print() ;
54
55     group b ;
56     b.age = 21 ;
57     strcpy(b.name,"debbie") ;
58     b.is_male = false ;
59     b.print() ;
60 }
```

Collection of
1) different types of data iems
2) Functions that operates on this data

Class: templates/cookie cutter/abstractions
No memory space

Encapsulation

Encapsulation: Bringing together data and functions into a single struture scope.

Advantage:
1. Objects no longer needs to be dependent on "outside" or "global functions" to alter its state or behaviour.
2. Objects receives messages (function calls) and act on these messages via its functions

Figure 6.2: Class

```
38 void code_with_classes() {
39     class group {
40     public:
41         int age ;
42         char name[10] ;
43         bool is_male ;
44         void print() {
45             cout << name << ";Age = " << age << ";Sex = " << ((is_male)? "male" : "female") << endl ;
46         }
47     };
48
49     group a ;
50     a.age = 17 ;
51     strcpy(a.name,"john") ;
52     a.is_male = true ;
53     a.print() ;
54
55     group b ;
56     b.age = 21 ;
57     strcpy(b.name,"debbie") ;
58     b.is_male = false ;
59     b.print() ;
60 }
```

Class: templates/cookie cutter/abstractions
No memory space

objects: Instances of the class
A cookie from a cookie cutter

Objects differ from each other only in the value of the data that they hold, but all of then has the same data fields of that class

Figure 6.3: Object

VASUDEVAMURTHY

6.3 *class* and objects

Figure 6.4: Class and objects

6.4 Accessing class members using dot and arrow operator

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c2.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
Definition of a class
----------------------------------------------------------------*/
class group_public {
public:
    int age ;
    char name[10] ;
    bool is_male ;
    void print() ;
};

/*----------------------------------------------------------------
Definition of a class
----------------------------------------------------------------*/
class group_private {
    int age ;
    char name[10] ;
    bool is_male ;
    void print() ;
};

/*----------------------------------------------------------------
    Accessing is using dot and arrow operators
----------------------------------------------------------------*/
static void accessing_public_elements_of_a_class() {
    group_public tom ;
    tom.age = 7 ;
    tom.is_male = true ;

    group_public* tom_ptr = &tom ;
    tom_ptr->age = 8 ;
    strcpy(tom_ptr->name,"TOM") ;
}

/*----------------------------------------------------------------
    Cannot accessing private members of the class
----------------------------------------------------------------*/
static void accessing_private_elements_of_a_class() {
    group_private tom ;
    //tom.age = 7 ;
   // error C2248: 'group_private::age' : cannot access private member declared in class 'group_private'
    //tom.is_male = true ;
    // error C2248: 'group_private::is_male' : cannot access private member declared in class 'group_private'
}

/*----------------------------------------------------------------

```


VASUDEVAMURTHY

```
----------------------------------------------------------------*/
int  main() {
   accessing_public_elements_of_a_class();
   accessing_private_elements_of_a_class();
   return 0 ;
}

```

VASUDEVAMURTHY

Figure 6.5: Public and private members of the class

VASUDEVAMURTHY

6.5 Accessing private class members

Accessing private members of the class

```
12 class group_private {
13 private:
14     int _age ;
15     char _name[10] ;
16     bool _is_male ;
17
18 public:
19     void set_age(int a) {_age = a ; }
20     int  age() const  { return _age ;}
21     void set_male() {_is_male = true ; }
22     void set_female() {_is_male = false ; }
23     bool is_male() const  { return _is_male ;}
24 };
```

DATA HIDING

age
name
is_male

We can change the way we store name

Changes data. Mutator functions

Inspectors. Only read data and does not modify . Const Accessor function

```
29 static void accessing_private_elements_of_a_class() {
30     group_private tom ;
31     //tom.age = 7 ;
32     // error C2248: 'group_private::age' : cannot access private member declared in class 'group_private'
33     //tom.is_male = true ;
34     // error C2248: 'group_private::is_male' : cannot access private member declared in class 'group_privat
35     tom.set_age(7) ;
36     tom.set_male() ;
37     cout << "Tom " << tom.age() << " " ;
38     cout << (tom.is_male()  ? "male" : "female") << endl ;
39 }
```

Accessing private members of the class using public interface

Principle of data hiding:
Ensures that users of the class are not able to manipulate the data members of the class directly

Figure 6.6: Accessing private class members and concept of data hiding

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c3.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
Definition of a class
----------------------------------------------------------------*/
class group_private {
private:
    int _age ;
    char _name[10] ;
    bool _is_male ;

public:
    void set_age(int a) {_age = a ; }
    int  age() const  { return _age ;}
    void set_male() {_is_male = true ; }
    void set_female() {_is_male = false ; }
    bool is_male() const  { return _is_male ;}
};

/*----------------------------------------------------------------
Accessing private members using public interface. Data hiding
----------------------------------------------------------------*/
static void accessing_private_elements_of_a_class() {
    group_private tom ;
    //tom.age = 7 ;
    // error C2248: 'group_private::age' : cannot access private member declared in class 'group_private'
    //tom.is_male = true ;
    // error C2248: 'group_private::is_male' : cannot access private member declared in class 'group_private'
    tom.set_age(7) ;
    tom.set_male() ;
    cout << "Tom " << tom.age() << " " ;
    cout << (tom.is_male()  ? "male" : "female") << endl ;
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
int  main() {
    accessing_private_elements_of_a_class();
    return 0 ;
}

```

VASUDEVAMURTHY

6.6 Separating interface and implementation

Figure 6.7: Interface and implementation

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c4.h
-----------------------------------------------------------------*/
#ifndef C4_H
#define C4_H

/*-----------------------------------------------------------------
Definition of the class - INTERFACE
-----------------------------------------------------------------*/
class group_private {
private:
    int _age ;
    char _name[10] ;
    bool _is_male ;

public:
    void set_age(int a) ;
    int  age() const ;
    void set_male() ;
    void set_female() ;
    bool is_male() const;
    void set_name(const char* s) ;
    const char* name() const;
    void print() const ;
};

#endif


```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c4.cpp
----------------------------------------------------------------*/

#include "c4.h"

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
  IMPLEMENTATION of the class
----------------------------------------------------------------*/

/*----------------------------------------------------------------
Implementation of set age
----------------------------------------------------------------*/
void group_private::set_age(int a) {
    _age = a ;
}

/*----------------------------------------------------------------
Implementation of age
----------------------------------------------------------------*/
int group_private::age() const {
    return _age ;
}

/*----------------------------------------------------------------
Implementation of set male
----------------------------------------------------------------*/
void group_private::set_male() {
    _is_male = true ;
}

/*----------------------------------------------------------------
Implementation of set female
----------------------------------------------------------------*/
void group_private::set_female() {
    _is_male = false ;
}

/*----------------------------------------------------------------
Implementation of is_male
----------------------------------------------------------------*/
bool group_private::is_male() const {
    return _is_male ;
}

/*----------------------------------------------------------------
Implementation of set name
----------------------------------------------------------------*/
void group_private::set_name(const char* s) {
    strcpy(_name,s) ;
}
```

```

/*-----------------------------------------------------------------
Implementation of get name
-----------------------------------------------------------------*/
const char* group_private::name() const {
    return _name ;
}

/*-----------------------------------------------------------------
Implementation of print
-----------------------------------------------------------------*/
void group_private::print() const {
    cout << name() << " " << age() << " " ;
    cout << (is_male()  ? "male" : "female") << endl ;
}


```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c4_main.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
----------------------------------------------------------------*/

#include "c4.h"

/*----------------------------------------------------------------
TOM JR 7 male
----------------------------------------------------------------*/
static void accessing_private_elements_of_a_class() {
   group_private tom ;
   //tom.age = 7 ;
   // error C2248: 'group_private::age' : cannot access private member declared in class 'group_private'
   //tom.is_male = true ;
   // error C2248: 'group_private::is_male' : cannot access private member declared in class 'group_private'
   tom.set_name("TOM JR") ;
   tom.set_age(7) ;
   tom.set_male() ;
   tom.print() ;
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
int  main() {
   accessing_private_elements_of_a_class();
   return 0 ;
}

```

VASUDEVAMURTHY

6.7 Allocating class objects on stack and heaps, phase 1

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c5.h
----------------------------------------------------------------*/

#ifndef C5_H
#define C5_H

/*----------------------------------------------------------------
Definition of the class
----------------------------------------------------------------*/
class student {
private:
    int _age ;
    char* _name;
    bool _is_male ;

public:
    void print() const ;
    void init(const char* name,int age,bool m=true) ;
    void fini() {delete [] _name;}
};

#endif

```

VASUDEVAMURTHY

```
12 class student {
13 private:
14     int _age ;
15     char* _name;
16     bool _is_male ;
17
18 public:
19     void print() const ;
20     void init(const char* name,int age,bool m=true) ;
21     void fini() {delete [] _name;}
22 };
23
```

```
52 static void allocating_on_stack() {
53     student tom ;
54     tom.init("Tom Jr",21) ;
55     tom.print() ;
56     student mary ;
57     mary.init("mary may",17,false) ;
58     mary.print() ;
59     student cs101[5] ;
60     for (int i = 0; i < 5; i++) {
61         char n[10] ;
62         strcpy(n,"student") ;
63         char p[2] ;
64         p[0] = i+'0' ;
65         p[1] = '\0' ;
66         strcat(n,p) ;
67         cs101[i].init(n,20+i,(i%2)?true:false) ;
68     }
69     for (int i = 0; i < 5; i++) {
70         cs101[i].print() ;
71         cs101[i].fini() ;
72     }
73     tom.fini() ;
74     mary.fini() ;
75 }
```

```
void student::init(const char* name,int age,bool m) {
    _name = new char[strlen(name)+1] ;
    strcpy(_name,name) ;
    _age = age ;
    _is_male = m ;
}
void fini() {delete [] _name;}
```

Figure 6.8: Allocating objects and arrays of objects on stack

VASUDEVAMURTHY

```
12  class student {
13  private:
14      int _age ;
15      char* _name;
16      bool _is_male ;
17
18  public:
19      void print() const ;
20      void init(const char* name,int age,bool m=true) ;
21      void fini() {delete [] _name;}
22  };
23
```

```
80  static void allocating_on_heap_1() {
81      student* tom = new student;
82      tom->init("Tom Jr",21) ;
83      tom->print() ;
84      student* mary = new student ;
85      mary->init("mary may",17,false) ;
86      mary->print() ;
87      student* cs101 = new student[5] ;
88      for (int i = 0; i < 5; i++) {
89          char n[10] ;
90          strcpy(n,"student") ;
91          char p[2] ;
92          p[0] = i+'0' ;
93          p[1] = '\0' ;
94          strcat(n,p) ;
95          cs101[i].init(n,20+i,(i%2)?true:false) ;
96      }
97      for (int i = 0; i < 5; i++) {
98          cs101[i].print() ;
99          cs101[i].fini() ;
100     }
101     delete [] cs101 ;
102     tom->fini() ;
103     mary->fini() ;
104     delete tom ;
105     delete mary ;
106 }
void student::init(const char* name,int age,bool m) {
    _name = new char[strlen(name)+1] ;
    strcpy(_name,name) ;
    _age = age ;
    _is_male = m ;
}
void init(const char* name,int age,bool m=true) ;
```

Figure 6.9: Allocating objects and arrays of objects on heap

VASUDEVAMURTHY

Figure 6.10: Allocating objects and arrays of objects on heap

VASU AMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c5.cpp
-----------------------------------------------------------------*/
#include "c5.h"

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
Implementation of print
-----------------------------------------------------------------*/
void student::print() const {
    cout << _name << " " << _age << " " ;
    cout << (_is_male  ? "male" : "female") << endl ;
}

/*-----------------------------------------------------------------
Implementation init
-----------------------------------------------------------------*/
void student::init(const char* name,int age,bool m) {
    _name = new char[strlen(name)+1] ;
    strcpy(_name,name) ;
    _age = age ;
    _is_male = m ;
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c5_main.cpp

 Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
-----------------------------------------------------------------*/
#include "c5.h"

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
size of the class student = 12
Tom Jr 21 male
mary may 17 female
student0 20 female
student1 21 male
student2 22 female
student3 23 male
student4 24 female
-----------------------------------------------------------------*/
static void allocating_on_stack() {
   student tom ;
   tom.init("Tom Jr",21) ;
   tom.print() ;
   student mary ;
   mary.init("mary may",17,false) ;
   mary.print() ;
   student cs101[5] ;
   for (int i = 0; i < 5; i++) {
      char n[10] ;
      strcpy(n,"student") ;
      char p[2] ;
      p[0] = i+'0' ;
      p[1] = '\0' ;
      strcat(n,p) ;
      cs101[i].init(n,20+i,(i%2)?true:false) ;
   }
   for (int i = 0; i < 5; i++) {
      cs101[i].print() ;
      cs101[i].fini() ;
   }
   tom.fini() ;
   mary.fini() ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void allocating_on_heap_1() {
   student* tom = new student;
   tom->init("Tom Jr",21) ;
   tom->print() ;
   student* mary = new student ;
   mary->init("mary may",17,false) ;
```


VASUDEVAMURTHY

```
   mary->print() ;
   student* cs101 = new student[5] ;
   for (int i = 0; i < 5; i++) {
      char n[10] ;
      strcpy(n,"student") ;
      char p[2] ;
      p[0] = i+'0' ;
      p[1] = '\0' ;
      strcat(n,p) ;
      cs101[i].init(n,20+i,(i%2)?true:false) ;
   }
   for (int i = 0; i < 5; i++) {
      cs101[i].print() ;
      cs101[i].fini() ;
   }
   delete [] cs101 ;
   tom->fini() ;
   mary->fini() ;
   delete tom ;
   delete mary ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void allocating_on_heap_2() {
   student* tom = new student;
   tom->init("Tom Jr",21) ;
   tom->print() ;
   student* mary = new student ;
   mary->init("mary may",17,false) ;
   mary->print() ;
   student** cs101 = new student*[5] ;
   for (int i = 0; i < 5; i++) {
      char n[10] ;
      strcpy(n,"student") ;
      char p[2] ;
      p[0] = i+'0' ;
      p[1] = '\0' ;
      strcat(n,p) ;
      cs101[i] = new student ;
      cs101[i]->init(n,20+i,(i%2)?true:false) ;
   }
   for (int i = 0; i < 5; i++) {
      cs101[i]->print() ;
   }
   for (int i = 0; i < 5; i++) {
      cs101[i]->fini() ;
      delete cs101[i];
   }
   delete [] cs101 ;
   tom->fini() ;
   mary->fini() ;
   delete tom;
   delete mary;
```


VASUDEVAMURTHY

```
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int  main() {
    cout << "size of the class student = " << sizeof(student) << endl ;
    allocating_on_stack();
    allocating_on_heap_1();
    allocating_on_heap_2();
    return 0 ;
}

```

VASUDEVAMURTHY

6.8 Shallow copying of objects, phase 1

Figure 6.11: Copying objects that has data on the heap

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c6.cpp

    FUM: Freeing unallocated memory
   Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
   THIS PROGRAM WILL CRASH
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
Definition of the class
-----------------------------------------------------------------*/
class student {
private:
   int _age ;
   char* _name;
   bool _is_male ;

public:
   void print() const ;
   void init(const char* name,int age,bool m=true) ;
   void fini() {delete [] _name;}
};

/*-----------------------------------------------------------------
Implementation of print
-----------------------------------------------------------------*/
void student::print() const {
   cout << _name << " " << _age << " " ;
   cout << (_is_male  ? "male" : "female") << endl ;
}

/*-----------------------------------------------------------------
Implementation init
-----------------------------------------------------------------*/
void student::init(const char* name,int age,bool m) {
   _name = new char[strlen(name)+1] ;
   strcpy(_name,name) ;
   _age = age ;
   _is_male = m ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void copy_crash() {
   student tom ;
   tom.init("Tom Jr",21) ;
   tom.print() ;
   student mary = tom ;
   cout << "finishing tom" << endl ;
   tom.fini() ;
```


VASUDEVAMURTHY

```
   cout << "finishing mary" << endl ;
   mary.fini() ;
}

/*-----------------------------------------------------------------
THIS PROGRAM WILL CRASH
-----------------------------------------------------------------*/
int  main() {
   cout << "size of the class student = " << sizeof(student) << endl ;
   copy_crash() ;
   return 0 ;
}

```

VASUDEVAMURTHY

6.9 Creating temporary objects on the stack, phase1

Figure 6.12: Creation of temporary objects due to pass by value and return by value

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c7.cpp

  Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
     Program exited with status code 0.
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
Definition of the class
-----------------------------------------------------------------*/
class student {
private:
    int _age ;
    char* _name;
    bool _is_male ;

public:
    void change_age(int i) {_age = i; }
    void print() const ;
    void init(const char* name,int age,bool m=true) ;
    void fini() {delete [] _name;}
};

/*-----------------------------------------------------------------
Implementation of print
-----------------------------------------------------------------*/
void student::print() const {
    cout << _name << " " << _age << " " ;
    cout << (_is_male  ? "male" : "female") << endl ;
}

/*-----------------------------------------------------------------
Implementation init
-----------------------------------------------------------------*/
void student::init(const char* name,int age,bool m) {
    _name = new char[strlen(name)+1] ;
    strcpy(_name,name) ;
    _age = age ;
    _is_male = m ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void pass_by_value(student t) {
    t.print() ;
    t.change_age(45) ;
    t.print() ;
}

/*-----------------------------------------------------------------
```


VASUDEVAMURTHY

```
Mary Sky 21 female
Mary Sky 45 female
Mary Sky 21 female
-----------------------------------------------------------------*/
static void pass_by_value_test() {
   student mary ;
   mary.init("Mary Sky",21,false) ;
   pass_by_value(mary) ;
   mary.print() ;
   mary.fini() ; //Will revisit when we write destructor
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static student return_by_value() {
   student t ;
   t.init("Bill Sr",23) ;
   return t ; //Will revisit when we write destructor
}

/*-----------------------------------------------------------------
Bill Sr 23 male
-----------------------------------------------------------------*/
static void return_by_value_test() {
   student k =  return_by_value();
   k.print() ;
   k.fini() ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int  main() {
   pass_by_value_test();
   return_by_value_test() ;
   return 0 ;
}

```

VASUDEVAMURTHY

6.10 inline function

An inline function is one whose code replaces the assembly language "CALL" that would be normally generated

All inline functions must be in class header files (*.h)

The compiler must have access to the source code inorder to do code substitution. That's why inline code must be in the same file

```
void set_age(int a) {_age = a ; }
int  age() const  {return _age ;}
```

Implicit inlining

```
inline bool group_private::is_male() const {
    return _is_male ;
}
```

Explicit inlining

If the inline request is NOT honored by compiler, the generated code will be larger, but the linker will NOT produce a duplicate definition error

Figure 6.13: Inlining of function

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c8.h
-----------------------------------------------------------------*/

#ifndef C8_H
#define C8_H
/*-----------------------------------------------------------------
Definition of a class
-----------------------------------------------------------------*/
class group_private {
private:
    int _age ;
    char _name[10] ;
    bool _is_male ;

public:
    void set_age(int a) {_age = a ; }
    int  age() const  {return _age ;}
    void set_male() ;
    void set_female();
    bool is_male() const ;
    void set_name(const char* s);
    const char* name() const ;
    void print() const ;
};

/*-----------------------------------------------------------------
Implementation of set male
-----------------------------------------------------------------*/
inline void group_private::set_male() {
    _is_male = true ;
}

/*-----------------------------------------------------------------
Implementation of set female
-----------------------------------------------------------------*/
inline void group_private::set_female() {
    _is_male = false ;
}

/*-----------------------------------------------------------------
Implementation of is_male
-----------------------------------------------------------------*/
inline bool group_private::is_male() const {
    return _is_male ;
}

/*-----------------------------------------------------------------
Implementation of get name
-----------------------------------------------------------------*/
inline const char* group_private::name() const {
    return _name ;
}
#endif
```


VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c8.cpp
-----------------------------------------------------------------*/
#include "c8.h"

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
Implementation of set name
-----------------------------------------------------------------*/
void group_private::set_name(const char* s) {
    strcpy(_name,s) ;
}

/*-----------------------------------------------------------------
Implementation of print
-----------------------------------------------------------------*/
void group_private::print() const {
    cout << name() << " " << age() << " " ;
    cout << (is_male()  ? "male" : "female") << endl ;
}

```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c8_main.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
     Program exited with status code 0.

----------------------------------------------------------------*/
#include "c8.h"

/*----------------------------------------------------------------
TOM JR 7 male
----------------------------------------------------------------*/
static void accessing_private_elements_of_a_class() {
   group_private tom ;
   //tom.age = 7 ;
   // error C2248: 'group_private::age' : cannot access private member declared in class 'group_private'
   //tom.is_male = true ;
   // error C2248: 'group_private::is_male' : cannot access private member declared in class 'group_private'
   tom.set_name("TOM JR") ;
   tom.set_age(7) ;
   tom.set_male() ;
   tom.print() ;
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
int  main() {
   accessing_private_elements_of_a_class();
   return 0 ;
}

```

VASUDEVAMURTHY

6.11 Understanding const

```
char* p = "jag"; //Non constant pointer, Non constant data
const char* p = "jag" ; //Non constant pointer, constant data
char* const p = "jag" ; //constant pointer,  non constant data
const char* const p = "jag" ; //constant pointer,  constant data
```

What's pointed to is constant	Pointer is constant
char *	p = "jag"
const char *	p = "jag"
char *	const p = "jag"
const char *	const p = "jag"

Use const whenever possible

Figure 6.14: Use **const** whenever possible

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: const.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
integer test
-----------------------------------------------------------------*/
static void int_test() {
  int x = 4 ;
  x = 10 ;
  //const int y = 2 ;
  //y = 89 ;
  //FAILS: error C3892: 'y' : you cannot assign to a variable that is const
}

/*-----------------------------------------------------------------
Generic print routine
-----------------------------------------------------------------*/
static void print(const char* title, const char* s) {
  cout << title << endl ;
  int l = strlen(s) ;
  for (int i = 0; i < l; i++) {
    cout << s[i] ;
  }
  cout << endl ;
}

/*-----------------------------------------------------------------
Non constant pointer Non constant data
-----------------------------------------------------------------*/
static void ncp_ncd() {
  char* p = new char[4] ;
  strcpy(p,"NPU") ;
  print("ncp_ncd BEFORE: ", p) ;
  p[0] = 'L' ;
  print("ncp_ncd AFTER: ", p) ;
  delete [] p ;
}

/*-----------------------------------------------------------------
Non constant pointer Constant data
-----------------------------------------------------------------*/
static void ncp_cd() {
  const char* p = "MIT" ;
  cout << "p =  " << p << endl ;
  //p[0] = 'N' ;
  //error C3892: 'p' : you cannot assign to a variable that is const
  const char* q = "UCSC" ;
  p = q ;
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Constant pointer Non constant data
-----------------------------------------------------------------*/
static void cp_ncd() {
  char* const p = new char[4] ;
  strcpy(p,"NPU") ;
  print("cp_ncd BEFORE: ", p) ;
  p[0] = 'L' ;
  print("cp_ncd AFTER: ", p) ;

  char* const p1 = new char[4] ;
  strcpy(p1,"MIT") ;
  print("cp_ncd BEFORE2: ", p1) ;
  //p = p1 ;
  // error C3892: 'p' : you cannot assign to a variable that is const
  delete [] p ;
  delete [] p1 ;
}

/*-----------------------------------------------------------------
Constant pointer Constant data
-----------------------------------------------------------------*/
static void cp_cd() {
  const char* const  p = "MIT" ;
  //p[0] = 'K' ;
  //error C3892: 'p' : you cannot assign to a variable that is const

  const char* q = "UCSC" ;
  //p = q ;
  //error C3892: 'p' : you cannot assign to a variable that is const
}

/*-----------------------------------------------------------------
Main
-----------------------------------------------------------------*/
int main() {
  int_test();
  cout << "_____________________________________" << endl ;
  ncp_ncd() ;
  cout << "_____________________________________" << endl ;
  ncp_cd() ;
  cout << "_____________________________________" << endl ;
  cp_ncd();
  cout << "_____________________________________" << endl ;
  cp_cd() ;
  cout << "_____________________________________" << endl ;
  return 0 ;
}

```

VASUDEVAMURTHY

6.12 Need for constructors

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: constdest_c_way.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
class book
-----------------------------------------------------------------*/
class book {
public:
    void init(const char *s, int c = 0, bool x = false);
    void finish() ;
    void print() const;

private:
    char* _name;
    int  _cost ;
    bool _in ;
};

/*-----------------------------------------------------------------
init function
-----------------------------------------------------------------*/
void book::init(const char *s, int c, bool x) {
    cout << "In book int " << s << endl ;
    _cost = c;
    _in = x ;
    int l = strlen(s) + 1 ;
    this->_name = new char[l] ;
    //you can write without this as: _name = new char[l] ;
    strcpy(_name,s) ;
}

/*-----------------------------------------------------------------
finish function
-----------------------------------------------------------------*/
void book::finish() {
    cout << "In book finish " << _name << endl ;
    delete [] this->_name ;
    //Without this: delete []_name ;
}

/*-----------------------------------------------------------------
print
-----------------------------------------------------------------*/
void book::print() const {
    cout << "Name " << _name << " Cost " << _cost ;
    (_in) ? cout <<" In " : cout <<  " Out " ;
    cout << endl ;
}

/*-----------------------------------------------------------------
```


VASUDEVAMURTHY

```
main

In book int algorithm
In book int C++
In book int VLSI design
Name algorithm Cost 0 Out
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
In book int A book on Java
Name A book on Java Cost 289 In
In book finish algorithm
In book finish C++
In book finish VLSI design
In book finish A book on Java
-----------------------------------------------------------------*/
int main() {
    book b1 ;
    b1.init("algorithm") ;
    book b2;
    b2.init("C++",79) ;
    book b3;
    b3.init("VLSI design", 168,true) ;
    b1.print() ;
    b2.print() ;
    b3.print() ;
    book* b4 = new book;
    b4->init("A book on Java",289,1)  ;
    b4->print() ;

    b1.finish() ;
    b2.finish() ;
    b3.finish() ;
    b4->finish() ;
    delete b4 ;
    return 0 ;
}

```

VASUDEVAMURTHY

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: constdest.cpp
-----------------------------------------------------------------*/
#include <iostream>
using namespace std;
#pragma warning(disable: 4996)
class book {
public:
  book(const char *s, int c = 0, bool x = false);
  ~book();
  void print() const;

private:
  char* _name;
  int  _cost;
  bool _in;
};

/*-----------------------------------------------------------------
Constructor
-----------------------------------------------------------------*/
book::book(const char *s, int c, bool x)
  : _name(nullptr), _cost(c), _in(x) {
  cout << "In book Constructor " << s << endl;

  auto l = strlen(s) + 1;
  this->_name = new char[l];
  //you can write without this as: _name = new char[l] ;
  strcpy(_name, s);

}

/*-----------------------------------------------------------------
Destructor
-----------------------------------------------------------------*/
book::~book() {
  cout << "In book Destructor " << _name << endl;
  delete[] this->_name;
  //Without this: delete []_name ;
}

/*-----------------------------------------------------------------
print
-----------------------------------------------------------------*/
void book::print() const {
  cout << "Name " << _name << " Cost " << _cost;
  (_in) ? cout << " In " : cout << " Out ";
  cout << endl;
```

```
}

/*-----------------------------------------------------------------
main

In book Constructor algorithm
In book Constructor C++
In book Constructor VLSI design
Name algorithm Cost 0 Out
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
In book Constructor A book on Java
Name A book on Java Cost 289 In
In book Destructor A book on Java
-----------------------------------------------------------------*/
void main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  //book b ; //error C2512: 'book': no appropriate default constructor available
  book b1("algorithm");
  book b2("C++", 79);
  book b3("VLSI design", 168, true);
  b1.print();
  b2.print();
  b3.print();
  book* b4 = new book("A book on Java", 289, 1);
  b4->print();
  delete b4;
}

```

Constructors

How do you know a constructor?

```
class obj {
        obj(<input parameters, may be null>)
};
```

1. Name of the constructor MUST BE name of the class
2. Constructor does NOT return any thing (not even void)
3. If you don't write a constuctor, compiler will give one which does nothing as follows:

```
class obj {
     obj() { }
}
```

4. There can be many constructors, same name but different parameters.
5. It cannot have the qualifiers like static or const
6. Constructors can be private

When constructors are called ?

A constuctor is automatically called when objects are created

```
Case 1: obj p ;
Case 2: obj *p = new obj(<input parameters>) ;
```

Creation of pointers, reference and primitive types does not call constructors

Figure 6.15: Constructors

VASUDEVAMURTHY

Destructor

How do you know a destructor?

```
class obj {
    ~obj() ;
}
```

1. The name of the destructor MUST BE name of the class preceded by ~
2. There are no inputs to destructor
3. Destructor does NOT return anything
4. If you don't write a destructor, compiler will give one with nothing inside
5. Destructor cannot be private

C++ guarantees that destructor is called when the object dies

When destructor is called?

Case 1: When an object goes out of scope

```
{
    obj p ;
    <may be some code here>
}
```

Case 2: When you call delete

```
{
    obj *p = new(obj) ;
    delete p ;
}
```

Destructor is NOT called when a pointers, reference or primitive type dies

Figure 6.16: Destructor

VASUDEVAMURTHY

6.13 constructors = default

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: constdestdefault.cpp
----------------------------------------------------------------*/
#include <iostream>

using namespace std;
#pragma warning(disable: 4996)
class book {
public:
  book() = default ;
  book(const char *s, int c = 0, bool x = false);
  ~book();
  void print() const;

private:
  char* _name = nullptr ;
  int  _cost = 55 ;
  bool _in = true ;
};

/*----------------------------------------------------------------
Constructor
----------------------------------------------------------------*/
book::book(const char *s, int c, bool x)
  : _name(nullptr), _cost(c), _in(x) {
  cout << "In book Constructor " << s << endl;

  auto l = strlen(s) + 1;
  this->_name = new char[l];
  //you can write without this as: _name = new char[l] ;
  strcpy(_name, s);

}

/*----------------------------------------------------------------
Destructor
----------------------------------------------------------------*/
book::~book() {
  cout << "In book Destructor " ;
  if (_name) {
    cout << _name ;
  }
  cout << endl ;
  delete[] this->_name;
  //Without this: delete []_name ;
}

/*----------------------------------------------------------------
```

```
print
-----------------------------------------------------------------*/
void book::print() const {
  if (_name) {
    cout << "Name " << _name ;
  }else {
    cout << "Name " << "NONAME" ;
  }
  cout << " Cost " << _cost;
  (_in) ? cout << " In " : cout << " Out ";
  cout << endl;
}

/*-----------------------------------------------------------------
main

Name NONAME Cost 55 In
In book Constructor algorithm
In book Constructor C++
In book Constructor VLSI design
Name algorithm Cost 0 Out
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
In book Constructor A book on Java
Name A book on Java Cost 289 In
-----------------------------------------------------------------*/
void main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  book b0;
  b0.print() ;
  book b1("algorithm");
  book b2("C++", 79);
  book b3("VLSI design", 168, true);
  b1.print();
  b2.print();
  b3.print();
  book* b4 = new book("A book on Java", 289, 1);
  b4->print();
  delete b4;
}
```

6.14 Creating temporary objects on the stack, phase 2

Figure 6.17: Creation of temporary objects due to pass by value and return by value

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c7m.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
Declartion of a class - INTERFACE
----------------------------------------------------------------*/
class student {
private:
    int _age ;
    char* _name;
    bool _is_male ;

public:
    student(const char* name,int age,bool m=true) ;
    ~student();
    void change_age(int i) {_age = i; }
    void print() const ;
};

/*----------------------------------------------------------------
Implementation of print
----------------------------------------------------------------*/
void student::print() const {
    cout << _name << " " << _age << " " ;
    cout << (_is_male  ? "male" : "female") << endl ;
}

/*----------------------------------------------------------------
Implementation of constructor
----------------------------------------------------------------*/
student::student(const char* name,int age,bool m):_name(0),_age(age),_is_male(m) {
    cout << "In student constructor " << name << ":" << age << ":" << boolalpha << m << endl ;
    _name = new char[strlen(name)+1] ;
    strcpy(_name,name) ;
}

/*----------------------------------------------------------------
Implementation of distructor
----------------------------------------------------------------*/
student::~student() {
    cout << "In student disstructor " << _name << ":" << _age << ":" << boolalpha << _is_male << endl ;
    delete [] _name;
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
static void pass_by_value(student t) {
    t.print() ;
    t.change_age(45) ;
    t.print() ;
}

/*----------------------------------------------------------------
Mary Sky 21 female
Mary Sky 45 female
Mary Sky 21 female
----------------------------------------------------------------*/
static void pass_by_value_test() {
    student mary("Mary Sky",21,false) ;
    pass_by_value(mary) ;
```

```
    mary.print() ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static student return_by_value() {
    student t("Bill Sr",23) ; ;
    return t ;
}

/*-----------------------------------------------------------------
Bill Sr 23 male
-----------------------------------------------------------------*/
static void return_by_value_test() {
    student k =  return_by_value();
    k.print() ;
}

/*-----------------------------------------------------------------
THIS PROGRAM WILL CRASH
-----------------------------------------------------------------*/
int  main() {
    pass_by_value_test();
    return_by_value_test() ;
    return 0 ;
}

```

VASUDEVAMURTHY

6.15 Shallow copying of objects, phase 2

Figure 6.18: Copying objects that has data on the heap

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:c6m.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
Declartion of a class - INTERFACE
-----------------------------------------------------------------*/
class student {
private:
    int _age ;
    char* _name;
    bool _is_male ;

public:
    student(const char* name,int age,bool m=true) ;
    ~student();
    void change_age(int i) {_age = i; }
    void print() const ;
};

/*-----------------------------------------------------------------
Implementation of print
-----------------------------------------------------------------*/
void student::print() const {
    cout << _name << " " << _age << " " ;
    cout << (_is_male  ? "male" : "female") << endl ;
}

/*-----------------------------------------------------------------
Implementation of constructor
-----------------------------------------------------------------*/
student::student(const char* name,int age,bool m) {
    cout << "In student constructor " << name << ":" << age << ":" << boolalpha << m << endl ;
    _name = new char[strlen(name)+1] ;
    strcpy(_name,name) ;
    _age = age ;
    _is_male = m ;
}

/*-----------------------------------------------------------------
Implementation of distructor
-----------------------------------------------------------------*/
student::~student() {
    cout << "In student disstructor " << _name << ":" << _age << ":" << boolalpha << _is_male << endl ;
    delete [] _name;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void copy_crash() {
```


VASUDEVAMURTHY

```
56    student tom("Tom Jr",21);
57    tom.print() ;
58    student mary = tom ;
59 }
60
61 /*-----------------------------------------------------------------
62 THIS PROGRAM WILL CRASH
63 -----------------------------------------------------------------*/
64 int  main() {
65    copy_crash() ;
66    return 0 ;
67 }
68
```

VASUDEVAMURTHY

6.16 Copy constructor and equal operator

VASUDEVAMURTHY

Figure 6.19: Deep copying of objects

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: copyeq_c_way.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

class book {
public:
    book() ;
    book(const char *s, int c = 0, bool x = false);
    ~book() ;
    void copy(const book& b) ;
    book& equal(const book& rhs) ;
    void print() const;
private:
    void _copy(const book& from) ;
    char* _name;
    int  _cost ;
    bool _in ;
    void _release();
};

/*-----------------------------------------------------------------
Constructor
-----------------------------------------------------------------*/
book::book(): _name(NULL),_cost(0),_in(0){
    cout << "book::book():" << endl ;
}

/*-----------------------------------------------------------------
Constructor
-----------------------------------------------------------------*/
book::book(const char *s, int c, bool x)
    : _name(NULL),_cost(c),_in(x){
        cout << "book::book(const char *s, int c, bool x): " << s << endl ;
        int l = strlen(s) + 1 ;
        _name = new char[l] ;
        strcpy(_name,s) ;
}

/*-----------------------------------------------------------------
Release heap memory of this class
-----------------------------------------------------------------*/
void book::_release() {
    delete [] _name ;
}

/*-----------------------------------------------------------------
Destructor
-----------------------------------------------------------------*/
book::~book() {
    cout << "In book Destructor " << _name << endl ;
    _release() ;
```

```
}

/*-----------------------------------------------------------------
Helper: copy function
-----------------------------------------------------------------*/
void book::_copy(const book& from) {
    int l = strlen(from._name) + 1 ;
    _name = new char[l] ;
    strcpy(_name,from._name) ;
    _cost = from._cost ;
    _in = from._in ;
}

/*-----------------------------------------------------------------
copy
-----------------------------------------------------------------*/
void book::copy(const book& b){
    cout << "In copy: " << b._name << endl ;
    _copy(b) ;
}

/*-----------------------------------------------------------------
equal
-----------------------------------------------------------------*/
book& book::equal(const book& rhs) {
    cout << "In equal: " << rhs._name << endl ;
    if (this != &rhs) {
        _release() ;
        _copy(rhs) ;
    }
    return *this ;
}

/*-----------------------------------------------------------------
print
-----------------------------------------------------------------*/
void book::print() const {
    cout << "Name " << _name << " Cost " << _cost ;
    (_in) ? cout <<" In " : cout <<  " Out " ;
    cout << endl ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
    book b1("algorithm") ;
    book b2("C++",79) ;
    book b3("VLSI design", 168,true) ;
    book b4 ;
    b4.copy(b2) ;
    book b5 ;
    b5.copy(b1) ;
    b1.print() ;
    b2.print() ;
```


VASUDEVAMURTHY

```
    b3.print() ;
    b4.print() ;
    b5.print() ;
    b1.equal(b3) ;
    b2.equal(b4) ;
    b1.print() ;
    b2.print() ;
    b3.print() ;
    b4.print() ;
    return 0 ;
}
/*-----------------------------------------------------------------
output of the above program
-----------------------------------------------------------------*/
/*
book::book(const char *s, int c, bool x): algorithm
book::book(const char *s, int c, bool x): C++
book::book(const char *s, int c, bool x): VLSI design
book::book():
In copy: C++
book::book():
In copy: algorithm
Name algorithm Cost 0 Out
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
Name C++ Cost 79 Out
Name algorithm Cost 0 Out
In equal: VLSI design
In equal: C++
Name VLSI design Cost 168 In
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
Name C++ Cost 79 Out
In book Destructor algorithm
In book Destructor C++
In book Destructor VLSI design
In book Destructor C++
In book Destructor VLSI design

*/
```

VASUDEVAMURTHY

Figure 6.20: Deep copying of objects using copy constructor and equal operator

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: copyeq.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

class book {
public:
   book(const char *s, int c = 0, bool x = false);
   ~book() ;
   book(const book& b) ;
   book& operator=(const book& rhs) ;
   void print() const;
private:
   void _copy(const book& from) ;
   char* _name;
   int  _cost ;
   bool _in ;
   void _release();
};

/*-----------------------------------------------------------------
Constructor
-----------------------------------------------------------------*/
book::book(const char *s, int c, bool x)
   : _name(NULL),_cost(c),_in(x){
      cout << "In book Constructor " << s << endl ;
      int l = strlen(s) + 1 ;
      _name = new char[l] ;
      strcpy(_name,s) ;
}

/*-----------------------------------------------------------------
Release heap memory of this class
-----------------------------------------------------------------*/
void book::_release() {
   delete [] _name ;
}

/*-----------------------------------------------------------------
Destructor
-----------------------------------------------------------------*/
book::~book() {
   cout << "In book Destructor " << _name << endl ;
   _release() ;
}

/*-----------------------------------------------------------------
Helper: copy function
-----------------------------------------------------------------*/
void book::_copy(const book& from) {
   int l = strlen(from._name) + 1 ;
   _name = new char[l] ;
```


VASUDEVAMURTHY

```
   strcpy(_name,from._name) ;
   _cost = from._cost ;
   _in = from._in ;
}

/*-----------------------------------------------------------------
Copy constructor
-----------------------------------------------------------------*/
book::book(const book& b){
   cout << "In copy Constructor " << b._name << endl ;
   _copy(b) ;
}

/*-----------------------------------------------------------------
equal operator
-----------------------------------------------------------------*/
book& book::operator=(const book& rhs) {
   cout << "In equal operator " << rhs._name << endl ;
   if (this != &rhs) {
      _release() ;
      _copy(rhs) ;
   }
   return *this ;
}

/*-----------------------------------------------------------------
print
-----------------------------------------------------------------*/
void book::print() const {
   cout << "Name " << _name << " Cost " << _cost ;
   (_in) ? cout <<" In " : cout <<  " Out " ;
   cout << endl ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
   book b1("algorithm") ;
   book b2("C++",79) ;
   book b3("VLSI design", 168,true) ;
   book b4(b2) ;
   book b5 = b1 ;
   b1.print() ;
   b2.print() ;
   b3.print() ;
   b4.print() ;
   b5.print() ;
   b1 = b3 ;
   b2 = b4 ;
   b1.print() ;
   b2.print() ;
   b3.print() ;
   b4.print() ;
   return 0 ;
```

```
}
/*-----------------------------------------------------------------
output of the above program
-----------------------------------------------------------------*/
/*
In book Constructor algorithm
In book Constructor C++
In book Constructor VLSI design
In copy Constructor C++
In copy Constructor algorithm
Name algorithm Cost 0 Out
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
Name C++ Cost 79 Out
Name algorithm Cost 0 Out
In equal operator VLSI design
In equal operator C++
Name VLSI design Cost 168 In
Name C++ Cost 79 Out
Name VLSI design Cost 168 In
Name C++ Cost 79 Out
In book Destructor algorithm
In book Destructor C++
In book Destructor VLSI design
In book Destructor C++
In book Destructor VLSI design

*/
```

VASUDEVAMURTHY

Copy Constructor

How do you know a Copy constructor?

```
class obj {
        obj(const obj& b)
};
```

1. Name of the copy constructor MUST BE name of the class
2. Copy constructor does NOT return any thing.
3. Note that the object from which we copy is a constant object passed by reference
4. If you don't write a copy constuctor, objects will be bit wised copied.

obj q(p)

without copy constructor

But we want like this:

Define a copy constructor and an assignment operator for classes with dynamically allocated memory

Figure 6.21: Copy constructor

VASUDEVAMURTHY

Figure 6.22: When copy constructor is called?

VASUDEVAMURTHY

Difference between copy constructor and equal operator

```
class obj {
      obj(const obj& b) ;
};
obj a ;
obj b(a) ;
obj c = a ;
```

```
class obj {
    obj& operator=(const obj& b) ;
};
obj a, b ;
b = a;
```

```
void book::_release() {
    delete [] _name ;
}
void book::_copy(const book& from) {
    int l = strlen(from._name) + 1 ;
    _name = new char[l+1] ;
    strcpy(_name,from._name) ;
    _cost = from._cost ;
    _in = from._in ;
}
```

Assign to all data members

```
book::book(const book& b){
    _copy(b) ;
}
```

```
book& book::operator=(const book& rhs) {
  if (this != &rhs) {
    _release() ;
    _copy(rhs) ;
  }
  return *this ;
}
```

Check for assignment to self in operator=

1. Copy constructor: No need to test to see if it is being initialized from itself.
2. Copy constructor: No need to release existing memory
3. Equal operator: A reference to the copied object is returned so that chaining can be done. EX; a = b = c = d ;

Have operator= return a reference to *this

Figure 6.23: Difference between copy constructor and equal operator

VASUDEVAMURTHY

6.16.1 Fraction class

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: c9.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
class fraction:
-----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
    cout << "In fraction constructor " << "(" << _numerator << "/" << _denominator << ")\n" ;
  }
  ~fraction() {
    cout << "In fraction  distructor " << "(" << _numerator << "/" << _denominator << ")\n" ;
    _numerator = 999999; _denominator=12345;
  }
  fraction mult(const fraction& b) {
    return fraction((this->_numerator * b._numerator),(this->_denominator * b._denominator)) ;
  }
  fraction mult1(const fraction& b) {
    fraction x(this->_numerator * b._numerator, this->_denominator * b._denominator) ;
    return x ;
  }
  friend ostream& operator<< (ostream& o, const fraction& a) {
    cout <<a. _numerator << "/" << a._denominator;
    return o ;
  }
  fraction(const fraction& a) {
    cout << "In copy constructor " << "(" << a._numerator << "/" << a._denominator << ")\n" ;
    _numerator = a._numerator;
    _denominator = a._denominator;
  }
  fraction& operator=(const fraction& rhs) {
    cout << "In Equal operator " << "(" << rhs._numerator << "/" << rhs._denominator << ")\n" ;
    if (this != &rhs) {
      _numerator = rhs._numerator;
      _denominator = rhs._denominator;
    }
    return *this ;
  }
  void change(int x, int y) {
    _numerator = x ;
    _denominator = y ;
  }
private:
  int _numerator ;
  int _denominator;
};

/*-----------------------------------------------------------------
To show:
creation of a calls copy constructor. a is a local object
Also, destructor is called on a when function ends -- local object deleted
-----------------------------------------------------------------*/
static void pass_by_value(fraction a) {
  //a is a local object. Constructor is called
  a.change(100,200) ;
  cout << "a in pass_by_value = " << a << endl ;
  //a is a local object. destructor is called and dies
  //change is never reflected to the external world
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
To show NO copy constructor is called. a is same as called object
Also, destructor NOT called when function ends
-----------------------------------------------------------------*/
static void pass_by_reference(fraction& a) {
  //a is reference object. No constructor is called
  a.change(100,200) ;
  cout << "a pass_by_reference = " << a << endl ;
  //a is reference. No destructor is called
}

/*-----------------------------------------------------------------
To show:
creation of t calls constructor -- local object
Copy constructor is called on t and copied object is returned to the caller
destructor is called on t at the end of function -- local object
-----------------------------------------------------------------*/
static fraction return_by_value(const fraction& a) {
  fraction t ; //local object
  t.change(100,200) ;
  cout << "t in return_by_value = " << t << endl ;
  return t ; //copy constructor called and copied object returned. local object t dies
}

/*-----------------------------------------------------------------
You are returning t to the caller which is dead
warning C4172: returning address of local variable or temporary
-----------------------------------------------------------------*/
static fraction& dangerous_return_by_alias(const fraction& a) {
  fraction t ; //local object
  t.change(100,200) ;
  cout << "t in dangerous_return_by_alias = " << t << endl ;
  return t ; //local object dies and you are returning reference to the dead object
}

/*-----------------------------------------------------------------
-----------------------------------------------------------------*/
int main() {
  fraction x(3,8) ;
  cout << "x = " << x << endl ;
  fraction y(x) ;
  cout << "y = " << y << endl ;
  fraction z = x ;
  cout << "z = " << z << endl ;
  z = y ;
  {
    fraction yx = y.mult(x) ;
    cout << "y * x = " << yx << endl ;
  }
  {
    fraction yx = y.mult1(x) ;
    cout << "y * x = " << yx << endl ;
  }
  {
    cout << "x = " << x << endl ;
    pass_by_value(x) ;
    cout << "x = " << x << endl ;

  }
  {
    cout << "x = " << x << endl ;
    pass_by_reference(x) ;
    cout << "x = " << x << endl ;

  }
  {
```

```
    cout << "y = " << y << endl ;
    fraction p = return_by_value(y) ;
    cout << "p = " << p << endl ;

  }
  {
    cout << "y = " << y << endl ;
    fraction p = dangerous_return_by_alias(y) ;
    cout << "p = " << p << endl ;

    cout << "y = " << y << endl ;
    fraction& q = dangerous_return_by_alias(y) ;
    cout << "q = " << q << endl ;
  }
  return 0 ;
}

```

VASUDEVAMURTHY

6.17 Private constructors

PRIVATE CONSTRUCTOR

```
class point {
private:
  double _x;
  double _y;
  static bool display;
  point::point(double x, double y) :_x(x), _y(y) {
    if (display) {
      cout << "In private constructor " << x << "  " << y << endl;
    }
  }
public:
  static point rectangular(double x, double y) {
    return point(x, y);
  }
  static point polar(double r, double theta) {
    //x = r*cos(theta) ;
    //y = r*sin(theta) ;
    double c = cos(theta * 3.14159265 / 180.00);
    double s = sin(theta * 3.14159265 / 180.00);
    return point(r*c, r*s);
  }
  friend ostream& operator<<(ostream& o, const point& p) {
    o << "x = " << p._x << " y = " << p._y;
    return o;
  }
};
```

```
int main() {
  //point p; //'point' : no appropriate default constructor
  point p1 = point::rectangular(12,5);
  cout << "rectangular(12,5) = " << p1 << endl;
  point p2 = point::polar(13,22.6);
  cout << "polar(13,22.6) = " << p2 << endl;
  return 0;
}
```

Figure 6.24: Need for private constructor

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
privateconstuctor.cpp
----------------------------------------------------------------*/
#include <iostream>
using namespace std;
#include <math.h>

/*----------------------------------------------------------------
Definition of the class
----------------------------------------------------------------*/
class point {
private:
  double _x;
  double _y;
  static bool _display;
  point::point(double x, double y) :_x(x), _y(y) {
    if (display()) {
      cout << "In private constructor " << x << "  " << y << endl;
    }
  }
public:
  static void setDisplay(bool x)  {
    _display = x;
  }
  static bool display() {
    return _display;
  }
  static point rectangular(double x, double y) {
    return point(x, y);
  }
  static point polar(double r, double theta) {
    //x = r*cos(theta) ;
    //y = r*sin(theta) ;
    double c = cos(theta * 3.14159265 / 180.00);
    double s = sin(theta * 3.14159265 / 180.00);
    return point(r*c, r*s);
  }
  friend ostream& operator<<(ostream& o, const point& p) {
    o << "x = " << p._x << " y = " << p._y;
    return o;
  }
};

/*----------------------------------------------------------------
Definition of the static variable
----------------------------------------------------------------*/
bool point::_display = true;

/*----------------------------------------------------------------
test
http://www.mathsisfun.com/polar-cartesian-coordinates.html
----------------------------------------------------------------*/
int main() {
  //point p; //'point' : no appropriate default constructor
  point::setDisplay(true);
  point p1 = point::rectangular(12,5);
  cout << "rectangular(12,5) = " << p1 << endl;
  point p2 = point::polar(13,22.6);
  cout << "polar(13,22.6) = " << p2 << endl;
  return 0;
}
```

VASUDEVAMURTHY

6.18 Constructor initialization list

6.18.1 Initialization versus assignment

```
11 class one {
12   public:
13     one(int x, int y) {
14       _x = x ;
15       _y = y ;
16     }
17
18   private:
19     int _x ;
20     int _y ;
21   };
```

Object value is assigned.
Inefficient, but OK

```
26 class two {
27   public:
28     two(int x, int y):_x(x),_y(y){}
29   private:
30     int _x ;
31     int _y ;
32   };
```

Object value is initialized

```
42 class three {
43   public:
44     three(int x, int y) {
45       _x = x ;
46       _y = y ;
47     }
48
49   private:
50     const int _x ;
51     int _y ;
52   };
```

Constant _x, Cannot be assigned
WILL NOT COMPILE

```
63 class three_reference {
64   public:
65     three_reference(int x, int y) {
66       _x = x ;
67       _y = y ;
68     }
69
70   private:
71     int& _x ;
72     int _y ;
73   };
```

Reference _x. Cannot be assigned
WILL NOT COMPILE

Initialization is mandatory for const and reference

```
79 class four {
80   public:
81     four(int x, int y):_x(x),_y(y){}
82   private:
83     const int _x ;
84     int _y ;
85   };
```

```
87 class four_reference {
88   public:
89     four_reference(int x, int y):_x(x),_y(y){}
90   private:
91     int& _x ;
92     int _y ;
93   };
94
```

Prefer initialization to assignment in constructors

Figure 6.25: Initialization versus assignment

6.18.2 Order of initialization

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: initalization.cpp
----------------------------------------------------------------*/
#include <iostream>
using namespace std;

/*----------------------------------------------------------------
When object one is created data is assigned, NOT initialized
----------------------------------------------------------------*/
class one {
public:
    one(int x, int y) {
        _x = x ;
        _y = y ;
    }

private:
    int _x ;
    int _y ;
};

/*----------------------------------------------------------------
When object two is created, data is initialized
----------------------------------------------------------------*/
class two {
public:
    two(int x, int y):_x(x),_y(y){}
private:
    int _x ;
    int _y ;
};

#if 0
/*----------------------------------------------------------------
When object three is created data is assigned, NOT initialized

WILL NOT compile
error C2758: 'three::_x' :
must be initialized in constructor base/member initializer list
----------------------------------------------------------------*/
class three {
public:
    three(int x, int y) {
        _x = x ;
        _y = y ;
    }
private:
    const int _x ;
    int _y ;
};
#endif

#if 0
/*----------------------------------------------------------------
```


VASUDEVAMURTHY

```
When object three is created data is assigned, NOT initialized

WILL NOT compile
error C2758: 'three_reference::_x' : must be initialized
in constructor base/member initializer list
----------------------------------------------------------------*/
class three_reference {
public:
   three_reference(int x, int y) {
      _x = x ;
      _y = y ;
   }
private:
   int& _x ;
   int _y ;
};
#endif

/*----------------------------------------------------------------
When object four is created, data is initialized
----------------------------------------------------------------*/
class four {
public:
   four(int x, int y):_x(x),_y(y){}
private:
   const int _x ;
   int _y ;
};

class four_reference {
public:
   four_reference(int x, int y):_x(x),_y(y){}
private:
   int& _x ;
   int _y ;
};

/*----------------------------------------------------------------
_size is guaranteed to be initialzed before _s
----------------------------------------------------------------*/
class five {
public:
   five(char *t):_size(strlen(t)),_s(new char[_size+1]) {
      strcpy(_s,t) ;
      cout << "string = " << _s << endl ;
   }
private:
   int _size ;
   char* _s ;
};

/*----------------------------------------------------------------
_s is guaranteed to be initialzed before _size
What _size it uses ???
----------------------------------------------------------------*/
```


VASUDEVAMURTHY

```
class six {
public:
   six(char *t):_size(strlen(t)), _s(new char[_size+1]) {
      strcpy(_s,t) ;
      cout << "string = " << _s << endl ;
   }
private:
   char* _s ;
   int _size ;
};

int main() {
   one u1(30,40) ;
   two u2(50,60) ;
   //three u3(67,99) ;
   //three_reference ur(8,89) ;
   four u4(6,8) ;
   four_reference u4r(6,8) ;
   five u5("Jagadeesh") ;
   six u6("ucsc extension at Santa Clara") ;
}
```

VASUDEVAMURTHY

Order of member initialization

1 2

```
98  class five {
99  public:
100     five(char *t):_size(strlen(t)),_s(new char[_size+1]) {
101       strcpy(_s,t) ;
102       cout << "string = " << _s << endl ;
103     }
104 private:
105     int _size ;
106     char* _s ;
107 };
```

1 Variable declartion order in class
2

Initialization order is tied to the order of the class layout

_size is guaranteed to be initialized before _s

```
113 class six {
114 public:
115     six(char *t):_size(strlen(t)), _s(new char[_size+1]) {
116       strcpy(_s,t) ;
117       cout << "string = " << _s << endl ;
118     }
119 private:
120     char* _s ;
121     int _size ;
122 };
```

If _s is initialized first, what _size it uses?

List members in an initialization list in the order in which they are declared

Figure 6.26: Order of initialization

VASU - VAMURTHY

6.19 Static member of a class

6.20 Size of a class

6.21 Need for friend

6.21.1 friend function

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```
/*----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: f1.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
class fraction:
----------------------------------------------------------------*/
class fraction {
public:
    fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
        cout << "In fraction constructor\n" ;
    }
    ~fraction() { cout << "In fraction  distructor\n" ;}
    int numerator() const { return _numerator ;}
    int denominator() const { return _denominator ;}

private:
    int _numerator ;
    int _denominator;
    fraction(const fraction& a);
    fraction& operator=(const fraction& rhs);
};

/*----------------------------------------------------------------
89
In fraction constructor
3/8
In fraction  distructor
----------------------------------------------------------------*/
int main() {
    int y = 89 ;
    cout << y << endl ;

    fraction x(3,8) ;
    //cout << x << endl ;  -- cout does not know fraction. It knows basic types only
    //no operator found which takes a right-hand operand of type 'fraction
    //cout << x._numerator ;// -- _numerator' : cannot access private member
    cout << x.numerator() ;
    cout << "/" ;
    cout << x.denominator()  ;
    cout << "\n" ;
    return 0 ;
}

```

VASUDEVAMURTHY

Operator << works on basic types

```
/*-----------------------------------------------------------------
class fraction:
-----------------------------------------------------------------*
class fraction {
public:
    fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
        cout << "In fraction constructor\n" ;
    }
    ~fraction() { cout << "In fraction  distructor\n" ;}
    int numerator() const { return _numerator ;}
    int denominator() const { return _denominator ;}

private:
    int _numerator ;
    int _denominator;
    fraction(const fraction& a);
    fraction& operator=(const fraction& rhs);
};
```

```
int y = 89 ;
cout << y << endl ;
```

1. cout is an object
2. cout has an overloaded operator
   cout << (basic_type)
   cout.operator<<(basic_type)

```
fraction x(3,8) ;
cout << x << endl ;
```

This won't work because we are calling
cout << (fraction)
cout does not know fraction

```
//cout << x << endl ;  -- cout does not know fraction. It knows basic types only
//no operator found which takes a right-hand operand of type 'fraction
//cout << x._numerator ;// -- _numerator' : cannot access private member
cout << x.numerator() ;
cout << "/" ;
cout << x.denominator()  ;
cout << "\n" ;
```

We need to take basic types from fraction and give to cout

Figure 6.27: Using cout basic types to print

VASUDEVAMURTHY

Consolidating fraction print

```
class fraction {
public:
    fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
        cout << "In fraction constructor\n" ;
    }
    ~fraction() { cout << "In fraction  distructor\n" ;}
    void print(ostream& o) const {
        o << _numerator ;
        o << "/" ;
        o << _denominator  ;
        o << "\n" ;
    }
```

```
fraction x(3,8) ;
x.print(cout) ;
```

It works, but different than basic types

```
int y = 98 ;
cout << y ;

fraction x(3,8) ;
x.print(cout) ;
```

Not consistent

Figure 6.28: Printing fraction object

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: f2.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
class fraction:
----------------------------------------------------------------*/
class fraction {
public:
    fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
        cout << "In fraction constructor\n" ;
    }
    ~fraction() { cout << "In fraction  distructor\n" ;}
    void print(ostream& o) const {
        o << _numerator ;
        o << "/" ;
        o << _denominator  ;
        o << "\n" ;
    }

private:
    int _numerator ;
    int _denominator;
    fraction(const fraction& a);
    fraction& operator=(const fraction& rhs);
};

/*----------------------------------------------------------------
89
In fraction constructor
3/8
In fraction  distructor
----------------------------------------------------------------*/
int main() {
    int y = 89 ;
    cout << y << endl ;
    fraction x(3,8) ;
    x.print(cout) ;
    return 0 ;
}

```

VASUDEVAMURTHY

Need for friend function

```
class fraction {
public:
    fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
        cout << "In fraction constructor\n" ;
    }
    ~fraction() { cout << "In fraction  distructor\n" ;}
    void print1(ostream& o, const fraction& a) {
        o << a. _numerator << "/" << a._denominator << endl ;
    }
    friend void print(ostream& o, const fraction& a) {
        o << a. _numerator << "/" << a._denominator << endl ;
    }
    //Exactly as print. print name is replaced by operator<<
    friend void operator<<(ostream& o, fraction& a) {
        o << a. _numerator << "/" << a._denominator << endl ;
    }

private:
    int _numerator ;
    int _denominator;
    fraction(const fraction& a);
    fraction& operator=(const fraction& rhs);
};
```

fraction x(3,8)

```
print1(cout,x) //What is print1? fails
x.print1(cout,x) ; //Works ugly

print(cout,x) ; //Works because of friend
//print is a NON member function
// Can be called WITHOUT an obj.print
//Must have obj as parameter in print function
```

instead of name: print
use: operator<<

Now everything works like basic types

```
int y = 89 ;
cout << y << endl ;

fraction x(3,8) ;
//print1(cout,x) ; //error C3861: 'print1': identifier not found
x.print1(cout,x) ; //Works, but ugly. User need to know print1
print(cout,x) ;  // Works without invoking from x
                 //But one of the parameter must be x
                 //Still user should know function name "print"
operator<<(cout,x) ; //infix notation
cout << x ; //postfix
```

Figure 6.29: Friend function

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: f3.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
class fraction:
-----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
    cout << "In fraction constructor\n" ;
  }
  ~fraction() { cout << "In fraction  distructor\n" ;}
  void print1(ostream& o) {
    o << _numerator << "/" << _denominator << endl ;
  }
  void print1(ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }
  friend void print(ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }
  //Exactly as print. print name is replaced by operator<<
  friend void operator<<(ostream& o, fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }

private:
  int _numerator ;
  int _denominator;
  fraction(const fraction& a);
  fraction& operator=(const fraction& rhs);
};

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  int a = 89 ;
  cout << "a = " << a << endl ;
  int b = 11 ;
  cout << "b = " << b << endl ;
  fraction x(3,8) ;
  x.print1(cout) ; // Works. Needs to invoke with x
  x.print1(cout,x) ; //Works. Needs to invoke with x
  //print1(cout,x) ; //error C3861: 'print1': identifier not found
  print(cout,x) ;  // invoke without x. One of the parameter must be x. User should know print
  operator<<(cout,x) ; //infix notation
  cout << x ; //postfix
  return 0 ;
}
```

VASUDEVAMURTHY

Need for friend function

```
class fraction {
public:
  fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
    cout << "In fraction constructor\n" ;
  }
  ~fraction() { cout << "In fraction  distructor\n" ;}
  void print1(ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }
  friend void print(ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }

  friend ostream& operator<< (ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator;
    return o ;
  }
```

fraction x(3,8)

print1(cout,x) //What is print1? fails
x.print1(cout,x) ; //Works ugly

print(cout,x) ; //Works because of friend
//print is a NON member function
// Can be called WITHOUT an obj.print
//Must have obj as parameter in print function

Chaining

Must have fraction as a parameter

Non member function

cout << x ;
operator<<(cout,x) ;

Note that we are invoking this function by cout(like print above)
and NOT by x. x is a parameter to cout function

Now everything works like basic types

```
int y = 98 ;
cout << y ;

fraction x(3,8) ;
fraction z(88,99) ;
cout << "x = " << x <<  " z =  " << z << endl ;
```

Figure 6.30: Friend function with chaining of cout

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: f4.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
class fraction:
----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
    cout << "In fraction constructor\n" ;
  }
  ~fraction() { cout << "In fraction  distructor\n" ;}
  void print1(ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }
  friend void print(ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator << endl ;
  }

  friend ostream& operator<< (ostream& o, const fraction& a) {
    o << a. _numerator << "/" << a._denominator;
    return o ;
  }

private:
  int _numerator ;
  int _denominator;
  fraction(const fraction& a);
  fraction& operator=(const fraction& rhs);
};

/*----------------------------------------------------------------

----------------------------------------------------------------*/
int main() {
  int y = 89 ;
  cout << y << endl ;

  fraction x(3,8) ;
  //print1(cout,x) ; //error C3861: 'print1': identifier not found
  x.print1(cout,x) ;
  print(cout,x) ;
  cout << x << endl ;
  fraction z(88,99) ;
  cout << "x = " << x << " z = " << z << endl ;
  return 0 ;
}

```

VASUDEVAMURTHY

Friend function

A non member function of the class

A function declared as friend in a class so that it has the same access as the class' members without having to be within the scope of the class.

Declaration

```
class cost {
  friend ostream& operator<<(ostream& o, const cost& c);
  private:
      int _base ;
};
```

Definition

```
ostream& operator<<(ostream& o, const cost& c){
   o << " base = " << c._base << endl ;
   return o ;
}
```

Usage

```
cost c ;
cout << c ;
operator<<(cout,c);
```

Note that this function is invoked by instance of cout and not by an instance of the user-defined class, cost

That means the function << must be declared as a binary friend of the class cost

1. The keyword friend is placed only in the function declaration of the friend function and not in the function definition.
2. A friend function, even though it is not a member function, would have the rights to access the private members of the class.
3. The function can be invoked without the use of an object
4. Friends are non-members hence do not get "this" pointer

Figure 6.31: Friend function

VASUDEVAMURTHY

6.21.2 friend class

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: friendclass.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

class patient ;

class data {
   int _age ;
   int _sugar ;
   int _bp ;
   friend class patient ; // Try commenting this line
};

class patient {
public:
   patient(int a, int s = 0, int b = 0);
   ~patient() ;
   void print(const char* p) const ;

private:
   data _d;
};

/*----------------------------------------------------------------
Constructor
----------------------------------------------------------------*/
patient::patient(int a, int s, int b){
   //Without friend class patient ;
   //error C2248: 'data::_sugar' : cannot access private member declared in class 'data'
   _d._age = a ;
   _d._sugar = s ;
   _d._bp = b ;
   cout << "In patient constructor\n" ;
}

/*----------------------------------------------------------------
Destructor
----------------------------------------------------------------*/
patient::~patient() {
   cout << "In patient Destructor \n";
}

/*----------------------------------------------------------------
print
----------------------------------------------------------------*/
void patient::print(const char *s) const {
   cout << s << " age = " << _d._age << " sugar = " <<_d._sugar << " bp = " << _d._bp << endl ;
}

/*----------------------------------------------------------------
main
```


VASUDEVAMURTHY

```
----------------------------------------------------------------*/
int main() {
    patient b1(25,90,118) ;
    b1.print("b1") ;
    return 0;
}
/*----------------------------------------------------------------
output of the above program
----------------------------------------------------------------*/
/*
In patient constructor
b1 age = 25 sugar = 90 bp = 118
In patient Destructor

*/

```

VASUDEVAMURTHY

6.22 Type conversions

```
          Primitive type to primitive type

static void  prim_2_prim() {
  {
    int a = 20 ;
    float f = a ; //conversion from 'int' to 'float', possible loss of data
    double d = 89.98 ;
    int ad = d ; //conversion from 'double' to 'int', possible loss of data
  }
  {
    int a = 20 ;
    float f = float(a) ; prim_type a = prim_type(other_prim_type b)
    double d = 89.98 ;
    int ad = int(d) ;
  }
}
```

Figure 6.32: Primitive to primitive type

VASUDEVAMURTHY

Primitive type to UDT

Happens through constructor

```
class Int {
public:
  Int(int x = 0):_d(x) {}
private:
  int _d ;
}

 static void  prim_2_udt() {
   int x = 89 ;
   Int I = Int(x) ;
 }
```

Figure 6.33: Primitive to UDT

VASUDEVAMURTHY

UDT to primitive type

Happens through
1. Member function
2. Conversion function

operator int() const { }

1. Name of the function must be operator followed by the type into which conversion is required
2. Must be a member function, not a friend
3. NO explicit return type. Function name itself is type
4. NO input argument to the function
5. Must return a value of the same type
6. Preferably a const function

```
class Int {
public:
  Int(int x = 0):_d(x) {cout << "In Int constructor " << x << endl ; }
  ~Int() { cout << "In Int destructor " << _d << endl ;}
  int getx() const {cout << "in getx() " << endl ;return _d ; }
  //conversion operator
  operator int() const {cout << "in int() " << endl ; return _d ; }
  operator float() const {cout << "in float() " << endl ; return float(_d) ; }
  operator const char*() const {cout << "in const char*() " << endl ; return "HAHA" ; }
}
 static void  udt2prim() {
   Int I = Int(89) ;
   int j = I.getx() ;
   cout << "j = " << j << endl ;
   int k = int(I) ;
   cout << "k = " << k << endl ;
   float f = float(I) ;
   cout << "f = " << f << endl ;
   const char* s = I ;
   cout << "s = " << s << endl ;
 }
```

```
In Int constructor 89
in getx()
j = 89
in int()
k = 89
in float()
f = 89
in const char*()
s = HAHA
In Int destructor 89
```

Figure 6.34: UDT to primitive

VASUDEVAMURTHY

Figure 6.35: UDT to UDT

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:type.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
forward declarations
----------------------------------------------------------------*/
//class Int ;
class Jstring;

/*----------------------------------------------------------------
Int class
----------------------------------------------------------------*/
class Int {
public:
  Int(int x = 0):_d(x) {cout << "In Int constructor " << x << endl ; }
  ~Int() { cout << "In Int destructor " << _d << endl ;}
  Int(const Int& r) { cout << "IN Int copy constructor  " << r._d << endl ; _d = r._d ; }
  Int& operator=(const Int& r) { cout << "IN Int equal operator " << r._d << endl ;  _d = r._d ; return *this ;}
  int getx() const {cout << "in getx() " << endl ;return _d ; }
  //conversion operator
  operator int() const {cout << "in int() " << endl ; return _d ; }
  operator float() const {cout << "in float() " << endl ; return float(_d) ; }
  operator const char*() const {cout << "in const char*() " << endl ; return "HAHA" ; }

  //Conversion operator that takes Int and returns Jstring
  //operator Jstring() const {cout << "in Int 2 Jstring conversion " << endl ;return Jstring("CONVERTED Jstring");}
  //You cannot do above:  error C2027: use of undefined type
  operator Jstring() const;
private:
  int _d ;
};

/*----------------------------------------------------------------
Jstring class
----------------------------------------------------------------*/
class Jstring {
public:
  Jstring(const char* x = 0):_d(x) {cout << "In Jstring constructor " << x << endl ; }
  ~Jstring() { cout << "In Jstring destructor " << _d << endl ;}
  Jstring(const Jstring& r) { cout << "IN Jstring copy constructor " << r._d << endl ; _d = r._d ; }
  Jstring& operator=(const Jstring& r) { cout << "IN Jstring equal operator " << r._d << endl ; _d = r._d ; return *this ;}
  const char* getx() const {cout << "in getx() " << endl ;return _d ; }
  //Conversion operator that takes Jstring and return Int
  operator Int() const { cout << "in Jstring 2 Int conversion " << endl ;return Int(420) ; }
private:
  const char* _d ;
};

/*----------------------------------------------------------------
Must be implemented after class definitions.
To overcome: error C2027: use of undefined type
----------------------------------------------------------------*/
Int::operator Jstring() const {cout << "in Int 2 Jstring conversion " << endl ;return Jstring("CONVERTED Jstring");}

/*----------------------------------------------------------------
Use C++ style casting:
type(variable)
```

```
----------------------------------------------------------------*/
static void  prim_2_prim() {
  {
    int a = 20 ;
    float f = a ; //conversion from 'int' to 'float', possible loss of data
    double d = 89.98 ; //conversion from 'double' to 'int', possible loss of data
    int ad = d ;
  }
  {
    int a = 20 ;
    float f = float(a) ;
    double d = 89.98 ;
    int ad = int(d) ;
  }
}

/*----------------------------------------------------------------
Use constructor

In Int constructor 89
In Int destructor 89
----------------------------------------------------------------*/
static void  prim_2_udt() {
  int x = 89 ;
  Int I = Int(x) ;
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
static void  udt2prim() {
  Int I = Int(89) ;
  int j = I.getx() ;
  cout << "j = " << j << endl ;
  int k = int(I) ; //explicit
  cout << "k = " << k << endl ;
  k = I ; //Compiler will figure out
  cout << "k = " << k << endl ;
  float f = float(I) ; //explicit
  cout << "f = " << f << endl ;
  f = I ; //Compiler will figure out
  const char* s = I  ;
  cout << "s = " << s << endl ;
}

/*----------------------------------------------------------------
In Int constructor 8
in getx()
j = 8
In Jstring constructor eight
in getx()
s = eight
in Jstring 2 Int conversion
In Int constructor 420
IN Int copy constructor  420
In Int destructor 420
cI = in int()
420
in Int 2 Jstring conversion
In Jstring constructor CONVERTED Jstring
IN Jstring copy constructor CONVERTED Jstring
In Jstring destructor CONVERTED Jstring
cS = in getx()
CONVERTED Jstring
In Jstring destructor CONVERTED Jstring
In Int destructor 420
```

```
In Jstring destructor eight
In Int destructor 8
-----------------------------------------------------------------*/
static void  udt2udt() {
  Int I = Int(8) ;
  int j = I.getx() ;
  cout << "j = " << j << endl ;
  Jstring S = Jstring("eight") ;
  const char*s = S.getx() ;
  cout << "s = " << s << endl ;
  Int cI = Int(S) ;
  cout << "cI = " << int(cI) << endl ;
  Jstring cS = Jstring(I) ;
  cout << "cS = " << cS.getx() << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  prim_2_prim();
  prim_2_udt();
  udt2prim();
  udt2udt() ;
  return 0 ;
}
```

VASUDEVAMURTHY

6.23 Usage of class string

6.24 Object composition

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: composition.cpp
----------------------------------------------------------------*/


/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include <iostream>
#include <fstream>

#include <iomanip>      // std::setprecision
using namespace std;

#ifdef WIN32
#include <cassert>
#include <ctime>
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#else
#include <assert.h>
#include <time.h>
#endif

/*----------------------------------------------------------------
Basic defines here
----------------------------------------------------------------*/
#define _NOCOPYOREQUAL_(X) X(const X& a);X& operator=(const X& a) ;
static const int SHOW = 1;

/*----------------------------------------------------------------
mstring class
----------------------------------------------------------------*/
class mstring{
public:
  mstring(const char* s = 0):_s(0) {
    if (SHOW) {
      cout << "in mstring constructor ";
      if (s) {
        cout << s ;
      }
      cout << endl ;
    }
    _allocate(s) ;
  }
  ~mstring() {
    if (SHOW) {
      cout << "in mstring distructor ";
      if (_s) {
        cout << _s ;
      }
      cout << endl ;
    }
    _free() ;
  }
  mstring(const mstring& rhs) {
    if (SHOW) {
      cout << "in mstring copy constructor " << rhs << endl ;
    }
    _allocate(rhs._s) ;
  }
  mstring& operator=(const mstring& rhs) {
    if (SHOW) {
      cout << "in mstring equal constructor " << rhs << endl ;
    }
    if (this != &rhs) {
```


VASUDEVAMURTHY

```
      _free() ;
      _allocate(rhs._s) ;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const mstring& rhs) {
    if (rhs._s) {
      o << rhs._s ;
    }
    return o ;
  }
  operator bool() const {return _s ? true: false;}
private:
  //data
  char* _s ;
  //functions
  void _allocate(const char* s) {_s = 0 ; if (s) {_s = new char[strlen(s)+1] ; strcpy(_s,s) ;}}
  void _free() {delete [] _s ; _s = 0; }
};

/*-----------------------------------------------------------------
SSN class
-----------------------------------------------------------------*/
class ssn{
public:
  ssn(const char* s, const char* n = 0 , const char* c = 0):_ssn(s),_name(n),_country(c){
    if (SHOW) {
      cout << "in ssn constructor " << endl ;
    }
  }
  ~ssn() {
    if (SHOW) {
      cout << "in ssn disstructor " << endl ;
    }
  }

  //COPY AND EQUAL OPERATOR NOT WRITTEN AS THIS CLASS HAS NO HEAP OBJECTS ON ITS OWN

  friend ostream& operator<<(ostream& o, const ssn& r) {
    //Why I am NOT doing this???
    //o << r._name << " " << r._ssn << " " << r._country;
    if (r._name) {
      o << r._name << " " ;
    }
    if (r._ssn) {
      o << r._ssn << " " ;
    }
    o << r._country; //I never put space for the last
    return o ;
  }

private:
  //data
  mstring _ssn ;
  mstring _name ; //SSN/Pan etc
  mstring _country ; //Country issued
};

/*-----------------------------------------------------------------
name class
-----------------------------------------------------------------*/
class name{
public:
  name(const char* f, const char* l = 0 , const char* m = 0, const char* t = 0):_fname(f),_lname(l),
  _mname(m),_title(t){
    if (SHOW) {
```

```
      cout << "in name constructor " << endl ;
    }
  }
  ~name() {
    if (SHOW) {
      cout << "in name disstructor " << endl ;
    }
  };

  //COPY AND EQUAL OPERATOR NOT WRITTEN AS THIS CLASS HAS NO HEAP OBJECTS ON ITS OWN

  friend ostream& operator<<(ostream& o, const name& r) {
    //Why I am NOT doing this???
    //o << r._title << " " << r._fname << " " << r._mname << " " << r._lname;
    if (r._title) {
      o << r._title << " " ;
    }
    if (r._fname) {
      o << r._fname << " " ;
    }
    if (r._mname) {
      o << r._mname << " " ;
    }
    o << r._lname; //I never put space for the last
    return o ;
  }

private:
  mstring _fname ;
  mstring _lname ;
  mstring _mname ;
  mstring _title ;
};

/*-----------------------------------------------------------------
date of birth class
-----------------------------------------------------------------*/
class dob{
public:
  dob(const int m , const int d, const int y):_month(m),_day(d),_year(y){
    if (SHOW) {
      cout << "in dob constructor " << endl ;
    }
  }
  ~dob(){
    if (SHOW) {
      cout << "in dob distructor " << endl ;
    }
  }

  //COPY AND EQUAL OPERATOR NOT WRITTEN AS THIS CLASS HAS NO HEAP OBJECTS ON ITS OWN

  friend ostream& operator<<(ostream& o, const dob& r) {
    o << r._month << " " << r._day << " " << r._year;
    return o ;
  }

private:
  //data
  unsigned int _month ;
  unsigned int _day ;
  unsigned int _year ;
};


/*-----------------------------------------------------------------
```


VASUDEVAMURTHY

```
person class
-----------------------------------------------------------------*/
class person{
public:
  person(const ssn& s, const name& n, const dob& d, const mstring& p, bool l = true):_ssn(s),_name(n),
  _dob(d),_current_place_of_residence(p),_isalive(l) {
    if (SHOW) {
      cout << "in person constructor " << endl ;
    }
  }
  ~person() {
    if (SHOW) {
      cout << "in person disstructor " << endl ;
    }
  }

  //COPY AND EQUAL OPERATOR NOT WRITTEN AS THIS CLASS HAS NO HEAP OBJECTS ON ITS OWN

  friend ostream& operator<<(ostream& o, const person& r) {
    o <<  r._name << " " << r._ssn << " " << r._dob << " " << r._current_place_of_residence ;
    if (!r._isalive) {
      o << " Dead " ;
    }
    return o ;
  }
private:
  //data
  ssn _ssn ;
  name _name ;
  dob _dob ;
  mstring _current_place_of_residence ;
  bool _isalive ;
};

/*-----------------------------------------------------------------
graduate class
-----------------------------------------------------------------*/
class graduate{
public:
  graduate(const person& p, const mstring& m, int y):_p(p),_degree_awarded(m),_year(y){
    if (SHOW) {
      cout << "in graduate constructor " << endl ;
    }
  }
  ~graduate() {
    if (SHOW) {
      cout << "in graduate disstructor " << endl ;
    }
  }

  //COPY AND EQUAL OPERATOR NOT WRITTEN AS THIS CLASS HAS NO HEAP OBJECTS ON ITS OWN

  friend ostream& operator<<(ostream& o, const graduate& r) {
    o << r._p << " " << r._degree_awarded << " " << r._year ;
    return o ;
  }
private:
  person _p ;
  mstring _degree_awarded ;
  int _year ;
};

/*-----------------------------------------------------------------
graduate class
-----------------------------------------------------------------*/
class list_of_graduates{
```


VASUDEVAMURTHY

```
public:
  list_of_graduates(int maxsize = 50):_maxsize(maxsize),_size(0),_list(0) {
    if (SHOW) {
      cout << "in list_of_graduates constructor " << endl ;
    }
    _list = new graduate* [_maxsize] ;
  }
  ~list_of_graduates() {
    if (SHOW) {
      cout << "in list_of_graduates disstructor " << endl ;
    }
    for (int i = 0; i < _size; ++i) {
      delete _list[i] ;
    }
    delete [] _list;
    _list = 0;
    _size = 0;
  }
  void append(const graduate& g) {
    assert(_size < _maxsize) ;
    graduate* t = new graduate(g) ;
    _list[_size++] = t ;
  }
  friend ostream& operator<<(ostream& o, const list_of_graduates& r) {
    for (int i = 0; i < r._size; ++i) {
      o << *(r._list[i]) << endl ;
    }
    return o ;
  }
private:
  int _maxsize ;
  int _size ;
  graduate** _list ;
  //DO NOT ALLOW TO COPY THIS OBJECT
  _NOCOPYOREQUAL_(list_of_graduates) ;
};

/*-----------------------------------------------------------------
test bench
-----------------------------------------------------------------*/
void test1() {
  ssn s1("657897","pan","India") ;
  cout << s1 << endl ;
  ssn s2("65-89-90","ssn","us") ;
  cout << s2 << endl ;

  ssn s3(s1) ;
  cout << s3 << endl ;
  s2 = s3 ;
  cout << s2 << endl ;
  cout << s3 << endl ;

  name n1("John") ;
  cout << n1 << endl ;
  name n2("John", "Smith") ;
  cout << n2 << endl ;
  name n3("John", "Smith", "Senior") ;
  cout << n3 << endl ;
  name n4("John", "Smith", "Senior", "Dr") ;
  cout << n4 << endl ;
}

/*-----------------------------------------------------------------
test bench
-----------------------------------------------------------------*/
void test2() {
```


VASUDEVAMURTHY

```
  list_of_graduates l(67) ;
  {
    ssn s("656-67-896","SSN","USA") ;
    name n("John", "Smith",0,"DR") ;
    dob d(8,14,1967) ;
    person p(s,n,d,"Singapore",false) ;
    graduate g(p,"Civil Engineering",2002) ;
    l.append(g) ;
  }
  {
    ssn s("JP9870","PAN","India") ;
    name n("Ram", "Rao","Subba") ;
    dob d(2,14,1977) ;
    person p(s,n,d,"USA") ;
    graduate g(p,"Doctor in Medicine",2020) ;
    l.append(g) ;
  }
  cout << l << endl ;
}

/*-----------------------------------------------------------------
Main
-----------------------------------------------------------------*/
int main() {
  test1() ;
  test2() ;
  return 0 ;
}

//EOF
```

VASUDEVAMURTHY

6.25 Returning member variables of a class

Returning member variable of the class

Question:
What is the significance of const return type from a function? Similar to:

```
const int func1();
```

Answer:

The const in this position guarantees that the caller of the function doesn't change the returned value.

Note that the use of const in a function's return value is more useful when the function returns a reference or a pointer to, say an object's data member.

It's less useful when return-by-value is used (as in example above) because the caller gets a local copy of the variable.

Therefore, there's not much point in limiting the use of the caller's own copy.

A function that returns its results by value should generally avoid const in the return type [FAQ 14.11]

Figure 6.36: Return by value should NOT use const

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename:t.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------

----------------------------------------------------------------*/
class salary {
public:
    salary(int s):_salary(s) {
        cout << "In salary constructor\n" ;
    }
    ~salary() {
        cout << "In salary destructor\n" ;
    }
    salary(const salary& s) {
        cout << "In salary copy constructor\n" ;
        _salary = s._salary;
    }
    salary& operator=(const salary& s) {
        cout << "In salary equal operator\n" ;
        _salary = s._salary;
        return *this ;
    }
    friend ostream& operator<<(ostream& o, const salary& s) {
        o << " Salary = " << s._salary << " " ;
        return o ;
    }
    void change_salary(int x) {
        _salary = x ;
    }
private:
    int _salary ;
};

/*----------------------------------------------------------------

----------------------------------------------------------------*/
class employee {
public:
    employee(int s, int id):_id(id),_s(s) {
        cout << "In employee constructor\n" ;
    }
    ~employee() {
        cout << "In employee destructor\n" ;
    }
    employee(const employee& e):_s(e._s),_id(e._id){
        cout << "In employee copy constructor\n" ;
    }
    employee& operator=(const employee& e) {
        _s = e._s ;
```


VASUDEVAMURTHY

```
        _id = e._id ;
        cout << "In employee equal operator\n" ;
        return *this ;
    }
    friend  ostream& operator<<(ostream& o, const employee& e) {
        o << "employee id = " << e._id << e._s ;
        return o ;
    }
    salary get_salary() const{
        return _s ;
    }
    void change_salary(int x) {
        _s.change_salary(x) ;
    }

private:
    salary _s ;
    int _id ;
};

/*-----------------------------------------------------------------
In salary constructor
In employee constructor
employee id = 7 Salary = 1000
In salary copy constructor
In salary destructor
employee id = 7 Salary = 1000
In employee destructor
In salary destructor
-----------------------------------------------------------------*/
int  main() {
    employee jim(1000,7) ;
    cout << jim << endl ;
    (jim.get_salary()).change_salary(2000) ;
    //Returned object is temp. Changing to 2000 will not change jim salary
    cout << jim << endl ;
    return 0 ;
}

```

VASUDEVAMURTHY

6.26 Problem set

Problem 6.26.1. Implement a **complex1 class**, as shown in the figure 6.37.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Implementing complex1 without C++ constructors and destructors

1. create a class called complex1 that has
    1. Real number, x >= 0
    2. Imaginary number, y >= 0
    3. construct a name x+iy and store the pointer.

2. You must create 3 files: complex1.h, complex1.cpp complex1test.cpp

3. You must use only util.h in complex1.h

4. You should be able to construct complex1 objects using 3 ways

```
complex1 h ;
h.init() ; //In that case x = 0, y = 0
h.init(5) ; //In that case x = 5, y = 0 ;
h.init(5,8) ; //In that case x = 5 and y = 8
```

5. Provide print, init and fini function as shown in c5.h

6. Test complex1 as shown in c5_main.cpp. You should allocate objects on stack and heap as shown in c5_main.cpp

7. print complex1test.cpp. Draw by hand, adjacent to each procedure, the allocation of objects as shown in figure 6.8, 6.9 and 6.10

8. e-mail all the 3 files (scan complex1test.cpp after hand drawing) to TA

Figure 6.37: Complex class(Phase 1)

VASUDEVAMURTHY

Problem 6.26.2. Implement a **integer matrix class**, as shown in the figure 6.38. You are given the test program in **intmatrix1test.cpp**.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

intmatrix1 (Phase1)

1. You cannot use constructors/destructors/copy constructor/equal operators
2. main program is given to you in intmatrix1test.cpp. You cannot change anything
3. Need to write intmatrix1.h and intmatrix1.cpp

Matrix initialization

```
intmatrix1 a ;   creates an empty matrix
a.init() ;         a = []

intmatrix1 b ;  creates a 3 x 4 matrix
b.init(3,4) ;    initialized to zero
                    0 0 0 0
                    0 0 0 0
                    0 0 0 0

intmatrix1 e ;  creates a 3 x 10 matrix
e.init(3,10,7) ;  initialized to 7
                 7 7 7 7 7 7 7 7 7 7
                 7 7 7 7 7 7 7 7 7 7
                 7 7 7 7 7 7 7 7 7 7

intmatrix1 c ;  creates an empty matrix
c.init(2,0,7) ;   as number of cols = 0

intmatrix1 d ;  creates an empty matrix
d.init(0,10,7) ;  as number of rows = 0

intmatrix1 f ;            creates a 2 x 3 matrix
f.init("1 2|3 4|5 6") ;    1 2
                           3 4
                           5 6

intmatrix1 g ;              creates a 2 x 3 matrix
g.init(" 1 2 |3 4 |5 6  ") ;  1 2
                              3 4
                              5 6

intmatrix1 h ;
h.init(" 1 2 |3 4 |5 6 8 ") ; creates an empty matrix
```

Never crashes. If not possible creates empty matrix

print

```
c.print(" Matrix c") ;

-------- Matrix c -----------
     Empty Matrix

g.print("Matrix g") ;

 -------- Matrix g --
 1 2
 3 4
 5 6
```

fini()

```
a.fini() ; //Free memory and clean
```

isEmpty()

Return true if matrix is empty
else return false

isEqual()

If matrix a == matrix b return true
else return false

add routine

1. Adds 2 matrices a and b(if possible)
2. Returns the answer(empty matrix if not possible)

mult routine

1. multiply 2 matrices a and b(if possible)
2. Returns the answer(empty matrix if not possible)

Figure 6.38: Integer matrix class(Phase 1)

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: intmatrix1test.cpp

On linux:
g++ intmatrix1.cpp intmatrix1test.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test intmatrix1 object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "intmatrix1.h"

/*-----------------------------------------------------------------
test init and fini
-----------------------------------------------------------------*/
void test_init_fini() {
  intmatrix1 a ;
  a.init() ;
  a.print("Matrix a") ;

  intmatrix1 b ;
  b.init(3,4) ;
  b.print("Matrix b") ;

  intmatrix1 c ;
  c.init(2,0,7) ;
  c.print("Matrix c") ;

  intmatrix1 d ;
  d.init(0,10,7) ;
  d.print("Matrix d") ;

  intmatrix1 e ;
  e.init(3,10,7) ;
  e.print("Matrix e") ;


  intmatrix1 f ;
  f.init("1 2|3 4|5 6") ;
  f.print("Matrix e") ;

  intmatrix1 g ;
  g.init(" 1 2 |3 4 |5 6  ") ;
  g.print("Matrix g") ;
  assert(f.isEqual(g)) ;
  assert(g.isEqual(f)) ;

  intmatrix1 h ;
  h.init(" 1 2 |3 4 |5 6 8 ") ;
  h.print("Matrix h") ;
  assert(h.isEmpty()) ;
  assert(!f.isEqual(h)) ;

  a.fini() ;
  b.fini() ;
  c.fini() ;
  d.fini() ;
  e.fini() ;
  f.fini() ;
```

VASUDEVAMURTHY

```
  g.fini() ;
}

/*-----------------------------------------------------------------
test add1
-----------------------------------------------------------------*/
void test_add1(const char* as, const char* bs, const char* anss) {
  intmatrix1 a ;
  a.init(as) ;
  a.print("Matrix a") ;
  intmatrix1 b ;
  b.init(bs) ;
  b.print("Matrix b") ;
  intmatrix1 s = a.add(b) ;
  s.print("matrix s") ;
  intmatrix1 ans;
  ans.init(anss) ;
  ans.print("matrix expected ans") ;
  assert(s.isEqual(ans)) ;
  assert(ans.isEqual(s)) ;
  a.fini() ;
  b.fini() ;
  s.fini() ;
  ans.fini() ;
}

/*-----------------------------------------------------------------
test add
-----------------------------------------------------------------*/
void test_add() {
  test_add1("7 9 11|13 15 17 "," 6 8 10| 12 14 16 ","13 17 21 | 25 29 33") ;
  test_add1("1 2 3|4 5  6 ","1 2  ","") ;
}

/*-----------------------------------------------------------------
test mult1
-----------------------------------------------------------------*/
void test_mult1(const char* as, const char* bs, const char* anss) {
  intmatrix1 a ;
  a.init(as) ;
  a.print("Matrix a") ;
  intmatrix1 b ;
  b.init(bs) ;
  b.print("Matrix b") ;
  intmatrix1 s = a.mult(b) ;
  s.print("matrix s") ;
  intmatrix1 ans;
  ans.init(anss) ;
  ans.print("matrix expected ans") ;
  assert(s.isEqual(ans)) ;
  assert(ans.isEqual(s)) ;
  a.fini() ;
  b.fini() ;
  s.fini() ;
  ans.fini() ;
  cout <<"---------------------------------\n" ;
}

/*-----------------------------------------------------------------
test mult
-----------------------------------------------------------------*/
void test_mult() {
  test_mult1("1 2 3"," 2 1 3 | 3 3 2| 4 1 2  ","20 10 13") ;
  test_mult1("3 4 2","13 9 7 15|8 7 4 6| 6 4 0 3 ","83 63 37 75") ;
  test_mult1("3","5 2 11|9 4 14","15 6 33|27 12 |42") ;
  const char* a = "3 9 0 2 2 9 5 2|0 2 2 1 9 6 6 8|7 5 6 1 4 9 8 9|3 3 2 9 2 1 7 4|1 9 0 1 2 9 5 2|4 2 ↙
```


VAI DEVAMURTHY

```
  0 3 7 3 9 1|5 9 0 6 6 7 8 2|9 3 4 6 8 4 9 1" ;
  const char* b = "6 1 6 0 8 3 0 0|6 8 9 0 6 6 7 2|4 8 2 0 5 4 6 7|2 4 4 2 2 6 9 8|4 8 2 2 4 6 4 1|1 5 ↙
  5 6 4 7 5 5|7 4 6 5 0 6 5 3|2 3 7 0 1 3 8 5";
  const char* s = "132 170 200  87 128 186 175 106|122 186 166  86  92 182 195 123|197 235 267 104 179 ↙
  243 253 178|128 140 164  63  86 162 194 140|118 164 184  85 110 174 166  98|138 142 144  83  91 162  ↙
  137  82|187 222 244 106 160 244 232 141|201 212 210  97 171 230 204 142";
  test_mult1(a,b,s) ;
  test_mult1("7 3|2 5 | 6 8| 9 0","8 14 0 3 1|7 11 5 91 3|8 4 19 5 57","") ;
}

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  test_init_fini();
  test_add() ;
  test_mult() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF
```

VASUDEVAMURTHY

Problem 6.26.3. Implement a **telephone_number class**, as shown in the figure 6.39.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

http://www.csgnetwork.com/phonenumcvtrev.html

INPUT : 1-800-4-XILINX
OUTPUT:1-800-4-945469

class telephone_number

1. The class should store
  1. The original phone number given by user and cannot be changed
  2. The converted phone numbers in terms of numbers. Convert only characters from {a ..z/A..Z}. All others should be left as is.
  That means numbers of characters before and conversions should be same
  3. You must use char*, not the STL string
  4.You must be able to copy and equal objects.
  5.The print routine should print both the original and converted string

You should write: telephone_number.h,
telephone_number.cpp
telephone_number_test.cpp

Test the program for various numbers. Also copy objects

Figure 6.39: telephone_number class

VASUDEVAMURTHY

Problem 6.26.4. Implement a **date class** using **Uspensky and Heaslet Algorithm**, as shown in the figure 6.40.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

http://mathforum.org/dr.math/faq/faq.calendar.html

Uspensky and Heaslet Algorithm

The computations used to derive the formula require the assumption that the year is greater than or equal to 1600. But we also need to be sure that the date involved refers to the Gregorian calendar which was invented in 1582 but not adopted by some countries until many years later.
The formula might not work for dates earlier than 1753.

```
t = (date + floor(2.6*m-0.2) + y + floor(y/4) + floor(c/4) -2*c) ;
day = (t % 7)
```

c = Century
y = Year
Note that c*100+y is the whole year
date = Date
m = month (starts from march)

In this algorithm, month starts from March
march=1, april =2, may=3,....dec = 10, jan=11, feb=12

Because of this rule, January and February are always counted as the 11th and 12th months of the previous year.

Aug 4 1914

date = 4
m = 8-2 = 6
1914
19*100+14=1914
c=19 y=14

t = 2
day = (2%7) = 2 =TUE

Jan 29 2064

date = 29
m = 11
2064 20*100+64 = 2064
c=20 y=64
Jan is previous year
y = 63
t =100
day = (100%7) = 2 = TUE

Mar 12 2011

date = 12
m = 1
C = 20
y = 11 20*100+11=2011

t = -8;
day = (-8%7) = 6 = SAT

day will be between 0 .. 6, which is encoded as follows:
0 = sunday 1 = monday 2 = tuesday 3 = wednesday
4 = thursday 5 = friday 6 = saturday

Once we have found t, we divide it by 7 and take the remainder.
Note that if the result for t is negative, care must be taken in calculating the proper remainder.
Suppose t = -17. When we divide by 7, we have to follow the same rules as for the greatest integer function; namely we find the greatest multiple of 7 less than -17, so the remainder will be positive (or zero).
-21 is the greatest multiple of 7 less than -17, so the remainder is 4 since -21 + 4 = -17.
Alternatively, we can say that -7 goes into -17 twice, making -14 and leaving a remainder of -3, then add 7 sin
the remainder is negative, so -3 + 7 is again a remainder of 4.

floor(x) = greatest integer less than or equal to x
floor(5.9) is 5
floor(5.2) is 5
floor(47) is 47
floor(-7.3) is -8
floor(-2) is -2

We need to find day such that day >= 0 && day <=6
-(7 * y) + day = -t
t=-17 -(7 *3) + 4 = -17 y is multiple of 7 and day=4
t=-8 -(7 *2) + 6 = -8 y is multiple of 2 and day=6

Figure 6.40: Computing **day** from **date**

VASUDEVAMURTHY

Problem 6.26.5. Design a program that simulates the movement of an elevator as show in the figure 6.41 and 6.42

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Simulation of the movement of an elevator

1. User should be able to specify the number of floors at the time of creation. Otherwise, assume that the building has 6 floors.
2. The elevators starts at floor 1.
3. Assuming the number of floors = 6, for the first time, flash the message as follows:

```
Constructing an elevator that has 6 floors
-------------------------------------------------------
You are In floor 1
Enter a floor number between 1 to 6
Enter 0 to close door or enter 7 to quit
-------------------------------------------------------
Enter:
```

4. At this point, user will enter the floor number. Your program must not crash for wrong inputs. Keep on getting data, until the user has presses 0 or 7 (number of floor +1) A typical data entry is as follows:

```
Enter:jdhfj
Enter:90-
Enter:-8
Enter:8
Enter:5
Will take to 5
Enter:3
Will
 take to 3
Enter:5
Will take to 5
Enter:0
```

5. When user enters 0, the door of the elevator will get closed. It will go up or down depending on the floors pressed. It will remain in the same floor, if no movement is required.

Continuing, the above example, the output is as follows:

Output the floor you are in.

Movement Direction

Prompt before opening door

Prompt all the buttons user has pressed already. This line should not appear if nothing is pressed

```
You are in floor 1. The door is closing....
Going up...........
Now you are in floor 3. The door is opening....
-------------------------------------------------------
You are In floor 3
Will take to 5
Enter a floor number between 1 to 6
Enter 0 to close door or enter 7 to quit
-------------------------------------------------------
Enter:
```

Keep on getting data until 0 or 7(Number of floors +1)

6. If user press 7(number of floors + 1), get out of the program and destroy all the resources. Output the following:

```
Enter:7
```

```
FIRE FIRE!. All get out of the elevator. Elevator will be destroyed
Elevator Destroyed
```

Figure 6.41: Simulating an elevator

VASUDEVAMURTHY

Implementing elevator movements

Must use the following files:
1. elevator.h
2. elevator.cpp
3. Must use the elevatortest.cpp given

You will write 3 class definitions in elevator.h

```
class scanner{
    //NO DATA
   public:
     void prompt_and_get(const char *title
       , unsigned int& i);
}
```

```
class button{
  public:
   //WRITE YOUR CODE HERE
  private:
    bool _b ;
};
```

```
class elevator{
  public:
    elevator(int n = 6) ;
    ~elevator() ;
   //NO OTHER FUNCTIONS HERE
  private:
    //WRITE ALL THE PRIVATE MEMBERS U NEED
    // TRY TO make scanner object static

    //WRITE ALL THE PRIVATE FUNCTIONS U NEED

    //YOU CANNOT REMOVE BELOW LINE
    _NOCOPYOREQUAL_(elevator) ;
};
```

elevatortest.cpp must have only these lines

```
#include "elevator.h"
void testbed() {
  elevator a;
  elevator b(100) ;
}
int main() {
    test_bed() ;
    return 0 ;
}
```

email: elevator.h and elevator.cpp.
capture the output of the program as a comment in elevator.cpp

Figure 6.42: Implementing an elevator

VASUDEVAMURTHY

Problem 6.26.6. Implement the game the **deal or no deal**.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Deal Or No Deal

You can play here

https://www.mysteinbach.ca/game-zone/245/deal-or-no-deal/

You can download game from

https://www.microsoft.com/en-us/store/p/deal-or-no-deal/9wzdncrfhvhd

Implement the above game in four flavors:

1. **Single interactive game**
2. **Single computer game (Use Random numbers)**
3. **Non Interactive: Play until you become a millionaire**
4. **Non Interactive: Play until you become a bankrupt(1 cent)**

Report after how many games you became a millionaire.
Report after how many games you became a bankrupt

Figure 6.43: Deal or no deal game

VASUDEVAMURTHY

Chapter 7

Operator overloading

7.1 Introduction

7.2 Overloading an operator as a member function or as a friend

VASUDEVAMURTHY

function as a member function (WITHOUT Friend)

```
class fraction {
public:
  fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
    cout << "In fraction constructor\n" ;
  }
  ~fraction() { cout << "In fraction  distructor\n" ;}
  fraction mult(const fraction& b) {
    return fraction((this->_numerator * b._numerator),(this->_denominator * b._denominator)) ;
  }
```

```
fraction x(3,8) ;
fraction y(1,2) ;

x.mult(y);  //OK
y.mult(x);  //OK
x.mult(5);  //OK
5.mult(x);  //NOT OK
fraction(5).mult(x) ; //OK
```

Where is this? ← 5.mult(x);

Figure 7.1: Function as a member function

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: friend1func.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
class fraction:
mult is NOT a friend
you can never do: 5 * fraction
----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1) :_numerator(n), _denominator(d) {
    cout << "In fraction constructor " << _numerator << "  " << _denominator << endl;
  }
  ~fraction() {
    cout << "In fraction  distructor " << _numerator << "  " << _denominator << endl;
  }
  fraction(const fraction& a) {
    cout << "In copy constructor " << a << endl;
    _numerator = a._numerator;
    _denominator = a._denominator;
  }

  fraction& operator=(const fraction& a) {
    cout << "In equal operator  " << a << endl;
    if (this != &(a)) {
      _numerator = a._numerator;
      _denominator = a._denominator;
    }
    return *this;
  }
  fraction mult(const fraction& b) const {
    // Construct an object
    // Call a copy constructor
    // Destroy the object
    // All the above are just done by just constructing the object to the caller
    if (0) {
      fraction t = fraction((this->_numerator * b._numerator), (this->_denominator * b._denominator));
      return t;
    } else {
      return fraction((this->_numerator * b._numerator), (this->_denominator *
```

```
          b._denominator));
      }
    }
    friend ostream& operator<< (ostream& o, const fraction& a) {
      o << a._numerator << "/" << a._denominator;
      return o;
    }
  private:
    int _numerator;
    int _denominator;
  };

  /*-----------------------------------------------------------------
  In fraction constructor
  x = 3/8
  In fraction constructor
  y = 1/2
  --------------1---------------
  In fraction constructor
   x * y = 3/16
  In fraction  distructor
  --------------2---------------
  In fraction constructor
  In fraction constructor
   x * 5 = 15/8
  In fraction  distructor
  In fraction  distructor
  --------------3---------------
  --------------4---------------
  In fraction constructor
  In fraction constructor
   5 * x = 15/8
  In fraction  distructor
  In fraction  distructor
  --------------5---------------
  In fraction  distructor
  In fraction  distructor

  -----------------------------------------------------------------*/
  int main() {
    fraction x(3, 8);
    cout << "x = " << x << endl;
    fraction y(1, 2);
    cout << "y = " << y << endl;
    cout << "--------------1---------------" << endl;
    {
      cout << " x * y = " << x.mult(y) << endl;
    }
    cout << "--------------2---------------" << endl;
```

function as a NON member function (Friend)

```
14 class fraction {
15 public:
16     fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
17         cout << "In fraction constructor\n" ;
18     }
19     ~fraction() { cout << "In fraction  distructor\n" ;}
20     friend fraction mult(const fraction& a, const fraction& b) {
21         return fraction((a._numerator * b._numerator),(a._denominator * b._denominator)) ;
22     }
```

```
fraction x(3,8) ;
fraction y(1,2) ;

mult(x,y);  //OK
mult(y,x);  //OK
mult(x,5);  //OK
mult(5,x);  //OK
```

There is NO this.

mult is a global function

Figure 7.2: Function as a non member friend function

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: friend2func.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
class fraction:
mult is friend
you can do: 5 * fraction
-----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1) :_numerator(n), _denominator(d) {
    cout << "In fraction constructor\n";
  }
  ~fraction() { cout << "In fraction  distructor\n"; }
  fraction(const fraction& a) {
    cout << "In fraction  copy constructor " << a << endl;
  }
  fraction& operator=(const fraction& rhs) {
    cout << "In fraction equal operator " << rhs << endl;
  }
  friend fraction mult(const fraction& a, const fraction& b) {
    // Construct an object
    // Call a copy constructor
    // Destroy the object
    // All the above are just done by just constructing the object to the caller
    return fraction((a._numerator * b._numerator), (a._denominator *
     b._denominator));
  }
  friend ostream& operator<< (ostream& o, const fraction& a) {
    o << a._numerator << "/" << a._denominator;
    return o;
  }
private:
  int _numerator;
  int _denominator;
};

/*-----------------------------------------------------------------
In fraction constructor
x = 3/8
In fraction constructor
y = 1/2
-----------1------------------
In fraction constructor
```

```
x * y = 3/16
In fraction  distructor
-----------2-------------------
In fraction constructor
In fraction constructor
x * 5 = 15/8
In fraction  distructor
In fraction  distructor
-----------3-------------------
In fraction constructor
In fraction constructor
5 * x = 15/8
In fraction  distructor
In fraction  distructor
-----------4-------------------
In fraction  distructor
In fraction  distructor
-----------------------------------------------------------------*/
int main() {
  fraction x(3, 8);
  cout << "x = " << x << endl;
  fraction y(1, 2);
  cout << "y = " << y << endl;
  cout << "-----------1-------------------" << endl;
  {
    cout << " x * y = " << mult(x, y) << endl;
  }
  cout << "-----------2-------------------" << endl;
  {
    cout << " x * 5 = " << mult(x, 5) << endl;
  }
  cout << "-----------3-------------------" << endl;
  {
    cout << " 5 * x = " << mult(5, x) << endl;
  }
  cout << "-----------4-------------------" << endl;
  return 0;
}
//EOF
```

Overloading * as a 1. member function
2. friend

```
class fraction {
public:
    fraction(int n = 0, int d = 1):_numerator(n),_denominator(d) {
        cout << "In fraction constructor\n" ;
    }
    ~fraction() { cout << "In fraction  distructor\n" ;}
    friend ostream& operator<< (ostream& o, const fraction& a) {
        cout <<a. _numerator << "/" << a._denominator;
        return o ;
    }

private:
    int _numerator ;
    int _denominator;
    fraction(const fraction& a);
    fraction& operator=(const fraction& rhs);
};
```

```
class fraction {
public
    fraction operator* (const fraction& b) {
        return fraction((this->_numerator * b._numerator),(this->_denominator * b._denominator)) ;
    }
};
```

x * y
y * x
x * 5
5 * x

Cannot do

NOTE that object has to be returned by value. It cannot be reference or a pointer

```
class fraction {
public
    friend fraction operator* (const fraction& a, const fraction& b) {
        return fraction((a._numerator * b._numerator),(a._denominator * b._denominator)) ;
    }
};
```

x * y
y * x
x * 5
5 * x

NOTE that object has to be returned by value. It cannot be reference or a pointer

Figure 7.3: Overloading operator *

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: friend1.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
class fraction:
operator* is NOT a friend
you can never do: 5 * fraction
-----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1) :_numerator(n), _denominator(d) {
    cout << "In fraction constructor\n";
  }
  ~fraction() { cout << "In fraction  distructor\n"; }
  fraction operator* (const fraction& b) {
    return fraction((this->_numerator * b._numerator), (this->_denominator *
     b._denominator));
  }
  friend ostream& operator<< (ostream& o, const fraction& a) {
    o << a._numerator << "/" << a._denominator;
    return o;
  }
  fraction(const fraction& a) {
    _numerator = a._numerator;
    _denominator = a._denominator;
    cout << "In fraction copy constructor\n";
  }
  fraction& operator=(const fraction& a) {
    _numerator = a._numerator;
    _denominator = a._denominator;
    cout << "In fraction EQUAL constructor\n";
  }

private:
  int _numerator;
  int _denominator;
};

/*-----------------------------------------------------------------
In fraction constructor
x = 3/8
In fraction constructor
y = 1/2
-------------------1---------------
```

```
In fraction constructor
 x * y = 3/16
In fraction  distructor
-------------------2---------------
In fraction constructor
In fraction constructor
 x * 5 = 15/8
In fraction  distructor
In fraction  distructor
-------------------3---------------
-------------------4---------------
In fraction constructor
In fraction constructor
In fraction constructor
In fraction constructor
            --------5---------------
In fraction constructor
In fraction constructor
In fraction constructor
 a * b * c * d = 8/8
In fraction  distructor
In fraction  distructor
In fraction  distructor
            --------6---------------
In fraction  distructor
In fraction  distructor
In fraction  distructor
In fraction  distructor
            --------7---------------
-------------------8---------------
In fraction  distructor
In fraction  distructor
-----------------------------------------------------------------*/
int main() {
  fraction x(3, 8);
  cout << "x = " << x << endl;
  fraction y(1, 2);
  cout << "y = " << y << endl;
  cout << "-------------------1---------------" << endl;
  {
    cout << " x * y = " << x * y << endl;
  }
  cout << "-------------------2---------------" << endl;
  {
    cout << " x * 5 = " << x * 5 << endl;
  }
  cout << "-------------------3---------------" << endl;
  {
    //cout << " 5 * x = " << 5 * x << endl ;
```

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: friend2.cpp
-----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*-----------------------------------------------------------------
class fraction:
operator* is a friend
you can do: 5 * fraction
-----------------------------------------------------------------*/
class fraction {
public:
  fraction(int n = 0, int d = 1) :_numerator(n), _denominator(d) {
    cout << "In fraction constructor\n";
  }
  ~fraction() { cout << "In fraction  distructor\n"; }
  fraction(const fraction& a) {
    cout << "In fraction copy constructor " << a << endl;
    _numerator = a._numerator;
    _denominator = a._denominator;
  }
  fraction operator=(const fraction& rhs) {
    cout << "In fraction = operator " << rhs << endl;
    if (this != &rhs) {
      _numerator = rhs._numerator;
      _denominator = rhs._denominator;
    }
    return *this;
  }
  friend fraction operator* (const fraction& a, const fraction& b) {
    //return fraction((a._numerator * b._numerator),(a._denominator * b._denominator)) ;
    fraction t((a._numerator * b._numerator), (a._denominator * b._denominator));
    return t;
  }
  friend ostream& operator<< (ostream& o, const fraction& a) {
    o << a._numerator << "/" << a._denominator;
    return o;
  }

private:
  int _numerator;
  int _denominator;
};

```

```
/*-----------------------------------------------------------------
In fraction constructor
x = 3/8
In fraction constructor
y = 1/2
-------------------1---------------
In fraction constructor
 x * y = 3/16
In fraction  distructor
-------------------2---------------
In fraction constructor
In fraction constructor
 x * 5 = 15/8
In fraction  distructor
In fraction  distructor
-------------------3---------------
In fraction constructor
In fraction constructor
 5 * x = 15/8
In fraction  distructor
In fraction  distructor
-------------------4---------------
In fraction constructor
In fraction constructor
In fraction constructor
In fraction constructor
            --------5---------------
In fraction constructor
In fraction constructor
In fraction constructor
 a * b * c * d = 8/8
In fraction  distructor
In fraction  distructor
In fraction  distructor
            --------6---------------
In fraction  distructor
In fraction  distructor
In fraction  distructor
In fraction  distructor
            --------7---------------
-------------------8---------------
In fraction  distructor
In fraction  distructor
-----------------------------------------------------------------*/
int main() {
  fraction x(3, 8);
  cout << "x = " << x << endl;
  fraction y(1, 2);
  cout << "y = " << y << endl;
```

7.3 Operators supported by C++ on basic types

VASUDEVAMURTHY

Arithmetic operators

Operator name		Syntax	Overloadable
Basic assignment		a = b	Yes
Addition		a + b	Yes
Subtraction		a - b	Yes
Unary plus (integer promotion)		+a	Yes
Unary minus (additive inverse)		-a	Yes
Multiplication		a * b	Yes
Division		a / b	Yes
Modulo (remainder)		a % b	Yes
Increment	Prefix	++a	Yes
	Suffix	a++	Yes
Decrement	Prefix	--a	Yes
	Suffix	a--	Yes

Comparison operators/Relational operato

Operator name	Syntax	Overloadable	I
Equal to	a == b	Yes	
Not equal to	a != b	Yes	
Greater than	a > b	Yes	
Less than	a < b	Yes	
Greater than or equal to	a >= b	Yes	
Less than or equal to	a <= b	Yes	

Logical operators

Operator name	Syntax	Overloadable
Logical negation (NOT)	!a	Yes
Logical AND	a && b	Yes
Logical OR	a || b	Yes

Bitwise operators

Operator name	Syntax	Overloadable
Bitwise NOT	~a	Yes
Bitwise AND	a & b	Yes
Bitwise OR	a | b	Yes
Bitwise XOR	a ^ b	Yes
Bitwise left shift[note 1]	a << b	Yes
Bitwise right shift[note 1]	a >> b	Yes

Compound-assignment operators

Operator name	Syntax	Overloadabl
Addition assignment	a += b	Yes
Subtraction assignment	a -= b	Yes
Multiplication assignment	a *= b	Yes
Division assignment	a /= b	Yes
Modulo assignment	a %= b	Yes
Bitwise AND assignment	a &= b	Yes
Bitwise OR assignment	a |= b	Yes
Bitwise XOR assignment	a ^= b	Yes
Bitwise left shift assignment	a <<= b	Yes
Bitwise right shift assignment	a >>= b	Yes

Figure 7.4: C++ operators on basic types

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Member and pointer operators

Operator name	Syntax	Overloadable
Array subscript	`a[b]`	Yes
Indirection ("variable pointed to by *a*")	`*a`	Yes
Reference ("address of *a*")	`&a`	Yes
Member *b* of object pointed to by *a*	`a->b`	Yes
Member *b* of object *a*	`a.b`	No
Member pointed to by *b* of object pointed to by *a*,	`a->*b`	Yes
Member pointed to by *b* of object *a*	`a.*b`	No

Other operators

Operator name	Syntax	Overloadable
Function call *See Function object.*	`a(a1, a2)`	Yes
Comma	`a, b`	Yes
Ternary conditional	`a ? b : c`	No
Scope resolution	`a::b`	No
Size-of	`sizeof(a)` `sizeof(type)`	No
Type identification	`typeid(a)` `typeid(type)`	No
Cast	`(type) a`	Yes
Allocate storage	`new type`	Yes
Allocate storage (array)	`new type[n]`	Yes
Deallocate storage	`delete a`	Yes
Deallocate storage (array)	`delete[] a`	Yes

Figure 7.5: C++ operators on basic types(contd)

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

7.4 Operators that cannot be overloaded

Figure 7.6: Operators that should not be overloaded

7.5 Operators precedence in C++

VASUDEVAMURTHY

C++ Operator Precedence

The operators at the top of this list are evaluated first. Operators within a group have the same precedence. All operators have left-to-right associativity unless otherwise noted.

Precedence	Operator	Description	Overloadable	Associativity
1	`::`	scope resolution	no	left to right
2	`()`	function call	yes	left to right
	`[]`	array access	yes	
	`->`	member access	yes	
	`.`		no	
	`++ --`	postfix	yes	
	`dynamic_cast static_cast reinterpret_cast const_cast`	type conversion	no	
	`typeid`	Get type information	no	
3	`! not`	logical negation	yes	right to left
	`~ compl`	bitwise negation (complement)	yes	
	`++ --`	prefix	yes	
	`+ -`	unary sign operations	yes	
	`* &`	indirection and reference	yes	
	`sizeof`	Size (of the type) of the operand in bytes	no	
	`new new[] delete delete[]`	dynamic memory management	yes	
	`(type)`	Cast to a given type	yes	
4	`->*`	member pointer selector	yes	left to right
	`.*`	member object selector	no	
5	`* / %`	arithmetic operations	yes	left to right
6	`+ -`			
7	`<< >>`	shift operations	yes	left to right
8	`< <= > >=`	relational operations	yes	left to right
9	`== != not_eq`			
10	`& bitand`	bitwise AND	yes	left to right
11	`^ xor`	bitwise XOR	yes	left to right
12	`| bitor`	bitwise OR	yes	left to right
13	`&& and`	logical AND	yes	left to right
14	`|| or`	logical OR	yes	left to right
15	`?:`	Ternary conditional (if-then-else)	no	right to left
16	`= += -= *= /= %= &= ^= |= <<= >>=`	assignment	yes	right to left
17	`,`	Sequential evaluation operator	yes	left to right

Figure 7.7: Operators precedence in C++

VASUDEVAMURTHY

7.6 Int class

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```
class Int {
    private:
        int _d ;
        char* _s;
        static bool _display ;
};

Int x(-234) ;
```

```
class Int {
    public:
        Int(int x = 0) ;
        ~Int() ;
        Int(const Int& x) ;
        Int& operator=(const Int& x) ;

        int num_digit() const { return strlen(_s) ; }
        const char& operator[] (int index) const ;
        friend ostream& operator<<(ostream& o, const Int& x) ;
        static void set_verbose(bool x) {_display = x ; }

        Int operator+() const ;
        Int  operator-() const;
        Int& operator++() ;
        Int  operator++(int i) ;
        Int& operator--() ;
        Int  operator--(int i) ;
        Int  operator~() const ;
        bool  operator!() const ;
        Int* operator&() ;

        /* binary operators as member functions */
        /*  + operator */
        Int operator+(const Int& b) const;
        Int operator+(int b)  const;

        /*  -= operator */
        Int& operator-=(const Int& b);
        Int& operator-=(int b);

        /* <= operator */
        bool operator<=(const Int& b) const ;
        bool operator<=(int b) const ;

        /* && operator */
        bool operator&&(const Int& b) const ;
        bool operator&&(int b) const ;
};
```

```
friend Int operator+(const Int& a) ;
friend Int  operator-(const Int& a);
friend Int& operator++(Int& a) ;
friend Int  operator++(Int& a,int i) ;
friend Int& operator--(Int& a) ;
friend Int  operator--(Int& a, int i) ;
friend Int  operator~(const Int& a) ;
friend bool  operator!(const Int& a) ;
friend Int* operator&(Int& a) ;

/* binary operators as friend functions */
/*  + operator */
friend Int operator+(const Int& a, const Int& b);
friend Int operator+(const Int& a, int b) ;
friend Int operator+(int a, const Int& b) ;

/*  -= operator */
friend Int& operator-=(Int& a, const Int& b);
friend Int& operator-=(Int& a, int b);

/* <= operator */
friend bool operator<=(const Int& a, const Int& b) ;
friend bool operator<=(const Int& a, int b) ;
friend bool operator<=(int b, const Int& a);

/* && operator */
friend bool operator&&(const Int& a, const Int& b);
friend bool operator&&(const Int& a, int b);
friend bool operator&&(int b, const Int& a);
```

Figure 7.8: Int class

VASUDEVAMURTHY

7.6.1 Unary operators

```
class Int{
private
  int _x ;
public:
  Int(int x = 0):_x(x) {
    cout << "In constructor\n" ;
  }
  ~Int() {
    cout << "In disstructor\n" ;
    _x = 999999 ;
  }
  Int(const Int& r) {
    _x = r._x ;
    cout << "In copy constructor\n" ;
  }
  Int& operator=(const Int& r) {
    _x = r._x ;
    cout << "In equal operatorr\n" ;
    return *this ;
  }
  friend ostream& operator<<(ostream& o, const Int& r) {
    o <<  r._x ;
    return o ;
  }
```

```
Int& pre_increment() {
  ++_x;
  return *this;
}
Int post_increment() {
    Int t = *this;
    ++_x;
    return t;
}
```

```
Int y(45);
Int z = y.pre_increment();
Int w = y.post_increment();
```

```
Int& increment() {
    ++_x;
    return *this;
}
Int increment(int junk) {
    Int t = *this;
    ++_x;
    return t;
}
```

```
Int y(45);
Int z = y.increment();
Int w = y.increment(9);
```

```
Int& operator++(){
  ++_x;
  return *this;
}
Int operator++(int j) {
  Int t = *this;
  ++_x;
  return t;
}
```

```
Int y(45);
Int z = ++y ;
Int w = y++ ;
```

Figure 7.9: Overloading unary increment

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: teachingincrement.cpp
----------------------------------------------------------------*/
#include <iostream>
using namespace std;

class Int {
public:
  Int(int x = 0) :_x(x) {
    cout << "In constructor\n";
  }
  ~Int() {
    cout << "In disstructor\n";
    _x = 999999;
  }
  Int(const Int& r) {
    _x = r._x;
    cout << "In copy constructor\n";
  }
  Int& operator=(const Int& r) {
    _x = r._x;
    cout << "In equal operatorr\n";
    return *this;
  }
  friend ostream& operator<<(ostream& o, const Int& r) {
    o << r._x;
    return o;
  }

  Int& pre_increment() {
    ++_x;
    return *this;
  }

  Int post_increment() {
    Int t = *this;
    ++_x;
    return t;
  }

  Int& increment() {
    ++_x;
    return *this;
  }

  Int increment(int junk) {
    Int t = *this;
    ++_x;
```

```
    return t;
  }

  Int& operator++() {
    ++_x;
    return *this;
  }
  Int operator++(int j) {
#if 0
    Int t = *this;
    ++_x;
    return t;
#endif
    return(_x++);
  }

private:
  int _x;

};

void main() {
  {
    Int y(45);
    cout << "y = " << y << endl;
    Int z = y.pre_increment();
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;

    cout << "y = " << y << endl;
    Int w = y.post_increment();
    cout << "y = " << y << endl;
    cout << "w = " << w << endl;
  }
  cout << "-------------------------------------\n";
  {
    Int y(45);
    cout << "y = " << y << endl;
    Int z = y.increment();
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;

    cout << "y = " << y << endl;
    Int w = y.increment(9);
    cout << "y = " << y << endl;
    cout << "w = " << w << endl;
  }
  cout << "-------------------------------------\n";
  {
```

VASUDEVAMURTHY

```
    Int y(45);
    cout << "y = " << y << endl;
    Int z = ++y;
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;

    cout << "y = " << y << endl;
    Int w = y++;
    cout << "y = " << y << endl;
    cout << "w = " << w << endl;
  }
  cout << "-------------------------------------\n";
}


```

VASUDEVAMS . HV

Unary operators

Operations with only one operators

1. Unary plus

```
Int x = 20 ;

Usage: Int y = +(x) ;
    Y is same as x
    x is unchanged

Int operator+() const ;
friend Int operator+(const Int& a) ;
```

Obj returned

x is unchanged

2. Unary minus

```
Int x = 20 ;

Usage: Int y =(-x) ;
   Y is negative of x
   x is unchanged

Int  operator-() const;
friend Int  operator-(const Int& a);
```

3. Pre increment

```
Int x = 20 ;

Usage: Int y= array[++x] ;
    x = 21 ;
    y = array[x] ;

Int& operator++() ;
friend Int& operator++(Int& a) ;
```

NONCONST
x is changed

x is changed

4. Post increment

```
Int x = 20 ;

Usage: Int y= array[x++] ;
    t = 20 ;
    x = 21 ;
    y = array[t] ;

Int  operator++(int i) ;
friend Int  operator++(Int& a,int i) ;
```

NONCONST
x is changed

obj(t) is returned

Dummy
unused
differentiate

5. Pre decrement

6. Post decrement

Figure 7.10: Overloading unary operators as member function

VASUDEVAMURTHY

Unary operators

Operations with only one operators

7. One's complement

```
int x = 0;
int y = ~x ; // x = 111111111111111 (32 ones)
```

y is one's complement of x
x is unchanged

```
Int  operator~() const ;
```

x is UNCHANGED

Returns object by value

8. Logical negation

```
int x = 23
!x returns false

int x = 0 ;
!x returns true
```

```
bool operator!() const ;
```

x is UNCHANGED

Returns true or false

9. Address

&x

```
Int* operator&() ;
```

Gives address to Int x object
x = 420
&x = 0038FDBC

```
friend Int* operator&(Int& a) ;
Int* operator&(Int& x){
   return &x ;
}
Int* _this() { return this; }
Int* operator&(Int& x){
   return x._this() ;
}
```

Infinite recursion

10. subscript []

```
const char& operator[] (int index) const ;
```

```
Int x = 8602
char c3 = x[3] = 2
```

X is UNCHANGED

Return char by const alias

Figure 7.11: Overloading unary operators as member function

VASUDEVAMURTHY

7.6.2 Binary operators

VASUDEVAMURTHY

Binary operators

Operations with two operators

Can be implemented as members or friends

1. + operator

```
a = 45 ;
b = 80 ;
a+b = 125 ;
```

a and b are passed by alias and untouched

```
friend Int operator+(const Int& a, const Int& b);
```

Must return answer by value -- chaining s = a+b+c+d ;

```
friend Int operator+(const Int& a, int b) ;
friend Int operator+(int a, const Int& b) ;
```

a + 3

3 + b

Int operator+(const Int& b) const;

Must return answer

a.operator+(b)
Same as: a + b
a and b cannot be changed. Hence const

a+b
b+a
a+5
~~5+a~~ Will not work

2. -= operator

```
a = 45 ;
b = 10 ;
```

a-=b
is same as a = a - b ;

a-=b = a = a -b = 35 ;
a gets changed. b is constant

b is unchanged. Hence const

Int& operator-=(const Int& b);

```
friend Int& operator-=(Int& a, const Int& b);
```

Must return answer by alias

a is changed. Hence NO CONST

```
friend Int& operator-=(Int& a, int b);
```

a = 45 ;

a + 5

Figure 7.12: Overloading binary operators as friend

VASUDEVAMURTHY

Binary operators

Operations with two operators

Can be implemented as members or friends

3. <= operator

```
a = 45 ;
b = 80 ;
a <= b is true
```

a and b are passed by alias and untouched

```
friend bool operator<=(const Int& a, const Int& b) ;
```

returns true or false

```
friend bool operator<=(const Int& a, int b) ;
friend bool operator<=(int b, const Int& a);
```

a <= 100

```
bool operator<=(const Int& b) const ;
```

3 <= b

4. && operator

```
a = 45 ;
b = 20 ;
a && b is true
```

```
bool operator&&(const Int& b) const ;
bool operator&&(int b) const ;
```

```
friend bool operator&&(const Int& a, const Int& b);
friend bool operator&&(const Int& a, int b);
friend bool operator&&(int b, const Int& a);
```

5. << operator

```
friend ostream& operator<<(ostream& o, const Int& x) ;
```

Chaining is possible

```
cout << x << y << z << endl ;
```

x cannot be changed and passed by alias

Figure 7.13: Overloading binary operators as friend

VASUDEVAMURTHY

7.6.3 Implementing Int class

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: IntTest.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
----------------------------------------------------------------*/
#include "Int.h"


/*----------------------------------------------------------------
Place 1 of 2 you have to change
----------------------------------------------------------------*/
//#define T int
#define T Int

/*----------------------------------------------------------------

----------------------------------------------------------------*/
static void P(const char* s) {
  cout << s << endl;
}

/*----------------------------------------------------------------
For exact match of x is Int
----------------------------------------------------------------*/
static void P(const char* s, const Int& x) {
  cout << s << " = " << x << endl;
}

/*----------------------------------------------------------------
For exact match of x is bool
----------------------------------------------------------------*/
static void P(const char* s, bool x) {
  cout << s << " = " << x << endl;
}

/*----------------------------------------------------------------
For exact match of x is int
----------------------------------------------------------------*/
static void P(const char* s, int x) {
  cout << s << " = " << x << endl;
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
static void test_unary() {
  P("unary + test");
  T x = 20;
```

```
  P("01. x", x);
  T yy = +x;
  P("02. +x", x);
  P("02a: yy", yy);
  P("unary - test");
  P("03. x", x);
  T y = -x;
  P("04.  x", x);
  P("04a.  y", y);
  y = -20;
  P("05. y", y);
  P("06. -y", (-y));
  P("pre increment ++x test");
  P("07. x", x);
  {
    T y = 0;
    P("08. y = ++x", y = ++x);
    P("08A. y", y);
  }
  P("09. x", x);
  P("post increment x++ test");
  P("10. x", x);
  {
    T y = 0;
    P("11. y = x++", y = x++);
    P("11A. y", y);
  }
  P("12. x", x);
  P("pre decrement --x test");
  P("13. x", x);
  P("14. --x", --x);
  P("15. x", x);
  P("post decrement x-- test");
  P("16. x", x);
  P("17. x--", x--);
  P("18. x", x);
  P("Complement ~x-- (one's complement) test");
  P("19. x", x);
  P("20. ~x", ~x);
  P("Logic negation  !x test");
  P("19. x", x);
  P("20. !x", !x);
  x = 0;
  P("21. x", x);
  P("22. !x", !x);
  x = 420;
  P("Address  &x test");
  P("23. x", x);
  cout << "24. &x = " << &x << endl;
```

```
  P("Indirection  *ptr test");
  T* ptr = &x;
  cout << "25. *ptr = " << *ptr << endl;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void test_binary() {
  P("Binary + test");
  T x = 20;
  T y = -10;
  P("01. x", x);
  P("02. y", y);
  P("03. x+y", x + y);
  P("04. x+5", x + 5);
#ifndef AS_MEMBER
  P("05. 5+x", 5 + x); //This will not work AS_MEMBER
#endif

  P("06. 5+8", 5 + 8);

  P("Binary - test");
  P("Binary * test");
  P("Binary / test");
  P("Binary % test");

  P("Binary & test");
  P("Binary | test");
  P("Binary ^ test");

  P("Binary += test");
  P("Binary -= test");
  P("07. x", x);
  P("08. y", y);
  P("09. x-=y ", x -= y);
  P("10. x", x);
  P("11. y", y);
  P("12. y-=x ", y -= x);
  P("13. x", x);
  P("14. y", y);
  P("15. x-=5 ", x -= 5);
  P("16. x", x);
  P("17. y", y);
  P("18. y-=3 ", y -= 3);

  P("Binary *= test");
  P("Binary /= test");
  P("Binary %= test");
```

VASUDEVAMURTHY

```

  P("Binary &= test");
  P("Binary |= test");
  P("Binary ^= test");

  P("Binary == test");
  P("Binary != test");
  P("Binary < test");
  P("Binary > test");

  P("Binary <= test");
  x = 67;
  y = 99;
  P("19. x", x);
  P("20. y", y);
  P("21. x<=y ", x <= y);
  P("22. x<=5 ", x <= 5);
#ifndef AS_MEMBER
  P("23. 5<=x ", 5 <= x);//This will not work AS_MEMBER
#endif

  P("Binary >= test");

  P("Binary && test");
  x = 67;
  y = 99;
  P("24. x", x);
  P("25. y", y);
  P("26. x&&y ", x&&y);
  P("27. x&&67 ", x && 67);
#ifndef AS_MEMBER
  P("23. 67&&x ", 66 && x);//This will not work AS_MEMBER
#endif

  P("Binary || test");

}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void test_special() {
  cout << "--------test_special----------" << endl;
  Int x(8602);
  cout << "x = " << x << " x[3] = " << x[3] - '0' << " x[100] = " << x[100] -
   '0' << endl;
  Int y(5);
  Int z(10);
  Int w = x + y + z + 77 + y;
```

VASUDEVAMURTHY

```
  cout << "w = " << w << endl;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  Int::set_verbose(true);
  test_unary();
  test_binary();
  test_special();
  return 0;
}

```

VASUDEVAN

7.7 A set class

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Figure 7.14: Set

VASUDEVAMURTHY

Figure 7.15: Set

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: iset64test.cpp

On linux:
g++ iset64.cpp iset64test.cpp
valgrind a.out

----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test iset64 object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "iset64.h"

/*----------------------------------------------------------------
test a set
----------------------------------------------------------------*/
void test_basic() {
  iset64 a;
  cout << "a = " << a << endl;
  a = a + 5;
  cout << "set a after adding 5 = " << a << endl;
  a = a + 5;
  cout << "set a after adding 5 = " << a << endl;
  a += 63;
  a += 0;
  cout << "set a after adding 0 and 63 = " << a << endl;
  int x[] = { 1, 3, 6 };
  iset64 b(x, sizeof(x) / sizeof(int));
  cout << "set b = " << b << endl;
  b = b - 3;
  cout << "set b after removing 3 = " << b << endl;
  b = b - 3;
  cout << "set b after removing 3 = " << b << endl;
  b = b - 10;
  cout << "set b after removing 10 = " << b << endl;
  b = b - 6;
  cout << "set b after removing 6 = " << b << endl;
  b = b - 1;
  cout << "set b after removing 1 = " << b << endl;
  b = b + 10;
  b = b + 2;
  cout << "set b after adding {10,2} = " << b << endl;
}

/*----------------------------------------------------------------
test union
----------------------------------------------------------------*/
void test_union() {
  {
    cout << "TESTING: iset64 operator+(const iset64& a, const iset64& b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 1;
    b += 2;
    b += 3;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    iset64 c = a + b;
```

```
    cout << "a + b = " << c << endl;
  }
  {
    cout << "TESTING:iset64 operator+(const iset64& a, const int b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << a << endl;
    a = a + 1;
    cout << "{1,2} + 1 = " << a << endl;
    a += 1;
    a += 2;
    cout << a << endl;
    a = a + 3;
    cout << "{1,2} + 3 = " << a << endl;
  }
  {
    cout << "TESTING:iset64 operator+(const int b, const iset64& a)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = 1 + a;
    cout << " 1 + {1,2} = " << a << endl;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = 3 + a;
    cout << " 3 + {1,2}  = " << a << endl;
  }

  {
    cout << "TESTING:iset64& iset64::operator+=(const iset64& a)" << endl;
    iset64  b;
    b += 1;
    b += 2;
    iset64 a;
    a += 1;
    a += 3;
    cout << "Set b " << b << endl;
    cout << "Set a " << a << endl;
    b += a;
    cout << " {1,2} + {1,3}  = " << b << endl;
  }
  {
    cout << "iset64& iset64::operator+=(const int b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a += 3;
    cout << " {1,2} + 3  = " << a << endl;
  }
  {
    //test chaining
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 3;
    b += 4;
    iset64 c;
    c += 7;
    c += 8;
    iset64 d = a + b + c + 5;
    cout << "Set a " << a << endl;
```

VASUDEVAMURTHY

```
    cout << "Set b " << b << endl;
    cout << "Set c " << c << endl;
    cout << "Set d " << d << endl;
  }
}

/*----------------------------------------------------------------
test difference
----------------------------------------------------------------*/
void test_difference() {
  {
    cout << "TESTING: iset64 operator-(const iset64& a, const iset64& b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 1;
    b += 2;
    iset64 c = a - b;
    cout << "Set a " << a << endl;
    cout << "Set b " << a << endl;
    cout << "a - b = " << c << endl;
  }
  {
    cout << "TESTING: iset64 operator-(const iset64& a, const iset64& b)" << endl;
    iset64 a;
    a += 1;
    a += 5;
    iset64 b;
    b += 1;
    b += 2;
    b += 3;
    iset64 c = a - b;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    cout << "a - b = " << c << endl;
  }

  {
    cout << "TESTING: iset64 operator-(const iset64& a, const int b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = a - 3;
    cout << "a - 3 = " << a << endl;
  }

  {
    cout << "TESTING: iset64 operator-(const int b, const iset64& a)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = 3 - a;
    cout << "3 - a = " << a << endl;
  }

  {
    cout << "TESTING: iset64& iset64::operator-=(const iset64& a)" << endl;
    iset64 a;
    a += 1;
    a += 3;
    iset64 b;
    b += 1;
    b += 2;
```


VALUDE VAMURTHY

```
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    b -= a;
    cout << "b -= a = " << b << endl;
  }

  {
    cout << "TESTING: iset64& iset64::operator-=(const int b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a -= 3;
    cout << "a -= 3 = " << a << endl;
  }
  {
    //test chaining
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 2;
    b += 4;
    iset64 c;
    c += 2;
    c += 8;
    iset64 d = a - b - c + 5;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    cout << "Set c " << c << endl;
    cout << "Set d " << d << endl;
  }
}

/*-----------------------------------------------------------------
test intersection
-----------------------------------------------------------------*/
void test_intersection() {
  {
    cout << "TESTING: iset64 operator*(const iset64& a, const iset64& b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 1;
    b += 2;
    b += 3;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    iset64 c = a * b;
    cout << "a * b = " << c << endl;
  }
  {
    cout << "TESTING:iset64 operator*(const iset64& a, const int b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = a * 1;
    cout << "{1,2} * 1 = " << a << endl;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = a * 3;
    cout << "{1,2} * 3 = " << a << endl;
  }
```

VASUDEVAMURTHY

```
  {
    cout << "TESTING:iset64 operator*(const int b, const iset64& a)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = 1 * a;
    cout << " 1 * {1,2} = " << a << endl;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a = 3 * a;
    cout << " 3 * {1,2}  = " << a << endl;
  }

  {
    cout << "TESTING:iset64& iset64::operator*=(const iset64& a)" << endl;
    iset64 b;
    b += 1;
    b += 2;
    iset64 a;
    a += 1;
    a += 3;
    cout << "Set b " << b << endl;
    cout << "Set a " << a << endl;
    b *= a;
    cout << " {1,2} * {1,3}  = " << b << endl;
  }
  {
    cout << "iset64& iset64::operator*=(const int b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    cout << "Set a " << a << endl;
    a *= 3;
    cout << " {1,2} * 3  = " << a << endl;
  }
  {
    //test chaining
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 2;
    b += 4;
    iset64 c;
    c += 2;
    c += 8;
    iset64 d = a * b * c + 5;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    cout << "Set c " << c << endl;
    cout << "Set d " << d << endl;
  }
}


/*-----------------------------------------------------------------
test equal
-----------------------------------------------------------------*/
void test_equal_not_equal() {
  {
    cout << "TESTING: bool operator==(const iset64& a, const iset64& b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
```


VAC JDEVAMURTHY

```
    iset64 b;
    b += 1;
    b += 2;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    cout << "a == b " << boolalpha << (a == b) << endl;
    b -= 1;
    cout << a;
    cout << b;
    cout << "a == b " << boolalpha << (a == b) << endl;
  }
  {
    cout << "TESTING: bool operator!=(const iset64& a, const iset64& b)" << endl;
    iset64 a;
    a += 1;
    a += 2;
    iset64 b;
    b += 1;
    b += 2;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    cout << "a != b " << boolalpha << (a != b) << endl;
    b -= 1;
    cout << "Set a " << a << endl;
    cout << "Set b " << b << endl;
    cout << "a != b " << boolalpha << (a != b) << endl;
  }
}

/*-----------------------------------------------------------------
++ and --
-----------------------------------------------------------------*/
void test_pre_post_inr_dec() {
  {
    int x[] = { 1, 2, 63 };
    iset64 a(x, sizeof(x) / sizeof(int));
    cout << "a = " << a << endl;
    ++a;
    cout << "++a = " << a << endl;
    int y[] = { 2, 3, 0 };
    iset64 b(y, sizeof(y) / sizeof(int));
    assert(a == b);
  }
  {
    int x[] = { 1, 2, 63 };
    iset64 a(x, sizeof(x) / sizeof(int));
    cout << "a = " << a << endl;
    iset64 acopy(x, sizeof(x) / sizeof(int));
    cout << "acopy = " << acopy << endl;
    iset64 rhs = a++;
    assert(rhs == acopy);
    cout << "a++ = " << a << endl;
    cout << "rhs = " << rhs << endl;
    int y[] = { 2, 3, 0 };
    iset64 b(y, sizeof(y) / sizeof(int));
    assert(a == b);
  }
  {
    int x[] = { 0,2,63 };
    iset64 a(x, sizeof(x) / sizeof(int));
    cout << "a = " << a << endl;
    --a;
    cout << "--a = " << a << endl;
    int y[] = { 63, 1, 62 };
    iset64 b(y, sizeof(y) / sizeof(int));
    assert(a == b);
```

```
  }
  {
    int x[] = { 0, 2, 63 };
    iset64 a(x, sizeof(x) / sizeof(int));
    cout << "a = " << a << endl;
    iset64 acopy(x, sizeof(x) / sizeof(int));
    cout << "acopy = " << acopy << endl;
    iset64 rhs = a--;
    assert(rhs == acopy);
    cout << "a-- = " << a << endl;
    cout << "rhs = " << rhs << endl;
    int y[] = { 63, 1, 62 };
    iset64 b(y, sizeof(y) / sizeof(int));
    assert(a == b);
  }
}

/*-----------------------------------------------------------------
~
Complement of a set.
The complement of A is the set of all element in the universal set U, but not in A.
a = {0,2,63}
x = ~a
{1,3,...,62}
-----------------------------------------------------------------*/
void test_complement() {
  {
    int x[] = { 0, 2, 63 };
    iset64 a(x, sizeof(x) / sizeof(int));
    cout << "a = " << a << endl;
    iset64 nota = (~a);
    cout << "~a = " << nota << endl;
    iset64 ans;
    ans += 1;
    for (int i = 3; i < 63; ++i) {
      ans += i;
    }
    cout << "ans = " << ans << endl;
    assert(nota == ans);
    ans = ~ans;
    cout << "~ans = " << ans << endl;
    assert(ans == a);
  }
}

/*-----------------------------------------------------------------
a = {0,2,63}
if (a) {

}
-----------------------------------------------------------------*/
void test_conversion_operator() {
  int x[] = { 0, 2, 63 };
  iset64 a(x, sizeof(x) / sizeof(int));
  cout << "a = " << a << endl;
  if (a) {
    cout << "a exists\n";
  } else {
    cout << "a does not exists\n";
  }
  iset64 b;
  cout << "b = " << b << endl;
  if (b) {
    cout << "b exists\n";
  } else {
    cout << "b does not exists\n";
```

```
  }
}

/*-----------------------------------------------------------------
a = {0,2,63}
if (!a) {

}
-----------------------------------------------------------------*/
void test_not_operator() {
  int x[] = { 0, 2, 63 };
  iset64 a(x, sizeof(x) / sizeof(int));
  cout << "a = " << a << endl;
  if (!a) {
    cout << "a does not exists\n";
  } else {
    cout << "a exists\n";
  }
  iset64 b;
  cout << "b = " << b << endl;
  if (!b) {
    cout << "b does not exists\n";
  } else {
    cout << "b exists\n";
  }
}

/*-----------------------------------------------------------------
(a+b)' = a'. b'
(a.b)' = a' + b'
-----------------------------------------------------------------*/
void test_demorgan_laws(const int x[], int lx, const int y[], int ly) {
  {
    iset64 a(x, lx);
    cout << "a = " << a << endl;

    iset64 b(y, ly);
    cout << "b = " << b << endl;

    iset64 aplusb = a + b;
    cout << "aplusb = " << aplusb << endl;

    iset64 aplusbbar = ~(aplusb);
    cout << "aplusbbar = " << aplusbbar << endl;

    iset64 abar = ~(a);
    cout << "abar = " << abar << endl;

    iset64 bbar = ~(b);
    cout << "bbar = " << bbar << endl;

    iset64 abarplusbbar = abar + bbar;
    cout << "abarplusbbar = " << abarplusbbar << endl;

    iset64 abardotbbar = abar * bbar;
    cout << "abardotbbar = " << abardotbbar << endl;

    iset64 adotb = a * b;
    cout << "adotb  = " << adotb << endl;

    iset64 adotbbar = ~(adotb);
    cout << "adotbbar  = " << adotbbar << endl;

    assert(aplusbbar == abardotbbar);
    cout << "Demorgan law (a+b)' = a'. b' is proved\n";
    assert(adotbbar == abarplusbbar);
```

```
    cout << "Demorgan law (a.b)' = a' + b' is proved\n";
  }
}

/*-----------------------------------------------------------------
(a+b)' = a'. b'
(a.b)' = a' + b'
-----------------------------------------------------------------*/
void test_demorgan_laws() {
  {
    int x[] = { 4, 5, 6 };
    int y[] = { 5, 6, 8 };
    test_demorgan_laws(x, (sizeof(x) / sizeof(int)), y, (sizeof(y) / sizeof(int)));
  }
  {
    int x[] = { 1,2,4,5 };
    int y[] = { 2,3,5,6 };
    test_demorgan_laws(x, (sizeof(x) / sizeof(int)), y, (sizeof(y) / sizeof(int)));
  }

}

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  test_basic();
  test_union();
  test_difference();
  test_intersection();
  test_equal_not_equal();
  test_pre_post_inr_dec();
  test_complement();
  test_conversion_operator();
  test_not_operator();
  test_demorgan_laws();
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed();
  return 0;
}

//EOF
```

VASUDEVAMURTHY

7.8 A long number class

unsigned longnum class

Write a class called longnum which can hold arbitrarly long unsigned numbers.

The class should have all the constructors, destructors, copy constructor equal operator,<<, and + operator

1. The user should be able to use this class like built-in type int object
2. One of the function in that class must be factorial function

```
class longnum {
   friend longnum factorial(unsigned n) ;
   void multiply(unsigned n) ;
};
```

Idea: You need to implement only + operator and this is trival.

Idea: Note that you need NOT have to implement * operator multiply function can be implemented as repeated +

factorial (100) should produce result like:

```
93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000
```

factorial(5) = 1 * 2 * 3 * 4 * 5 = 120

Idea:

```
longnum factorial(unsigned n) {
   longnum o(1) ;
   for(unsigned i = 2; i <= n ; i++) {
      o.multiply(i) ;
   }
   return o ;
}
```

Use this main program like this to test your class.

```
int main() {
   longnum a(789) ;
   cout << "a = " <<  a << endl ;
   longnum b("56789") ;
   cout << "b = " <<  b << endl ;
   longnum c("123456789012345678901234567890123456789012345678901234567890") ;
   cout << "c = " <<  c << endl ;
   longnum ta(a) ;
   cout << "ta = " << ta << endl ;
   ta = b ;
   cout << "ta = " << ta << endl ;
   cout << "a + c = " << a + c << endl ;
   a.multiply(5) ;
   cout << "a * 5 = " << a << endl ;
   longnum f100 = factorial(100) ;
   cout << "Factorial of 100 = " << endl ;
   cout << f100 << endl ;
   longnum f1000 = factorial(1000) ;
   cout << "Factorial of 1000 = " << endl ;
   cout << f1000 << endl ;
   return 0 ;
}
```

```
a = 789
b = 56789
c = 123456789012
1234567890123456
ta = 789
ta = 56789
a = 789 a + 20 =
...........
Factorial of 100=
.........
```

Figure 7.16: A long number class

VASUDEVAMURTHY

7.9 mstring class

mstring class behaves exactly like **stl string** class. **mstringtest.cpp** file is given to you. You need to implement all the functions required in the **mstringtest.cpp**. Then comment the line **#define mstring string** in **mstringtest.cpp**. Your program must pass all the **asserts** and there should be no **memory leaks**. You should write all the declarations of the class **mstring** in **mstring.h** and all definitions of the class **mstring** in **mstring.cpp**. You cannot alter anything in **mstringtest.cpp**, except commenting **#define mstring string**. **class mstring** in the file **mstring.h** must use **char* _s** as the **ONLY private data member**. You cannot use any other data members. You are free to use any number of private and public functions. **email** me only **mstring.h** and **mstring.cpp**

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: mstringtest.cpp

On linux:
g++ mstring.cpp mstringtest.cpp
valgrind a.out
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test mstring object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "mstring.h"

/*----------------------------------------------------------------
COMMENT THE LINE BELOW AFTER U WRITE mstring class
----------------------------------------------------------------*/
#define mstring string

/*----------------------------------------------------------------
Assert strings are equivalent
----------------------------------------------------------------*/
static void assert_string_mstring_equal(const string& s, const mstring& m) {
  assert(strcmp(s.c_str(),m.c_str()) == 0) ;
}

/*----------------------------------------------------------------
Assert strings are not equivalent
----------------------------------------------------------------*/
static void assert_string_mstring_not_equal(const string& s, const mstring& m) {
  assert(strcmp(s.c_str(),m.c_str())) ;
}

/*----------------------------------------------------------------
test constructors
----------------------------------------------------------------*/
void test_constructors() {
  {
    string gs ; //gold
    mstring s ;
    assert(gs.empty()) ;
    assert(s.empty()) ;
    assert(gs.size() == 0) ;
    assert(s.size() == 0) ;
    assert(gs.length() == 0) ;
    assert(s.length() == 0) ;
    assert(gs.length() == s.length()) ;
  }
  {
    string gs("Pack my box with five dozens liquor jugs") ; //gold
    mstring s("Pack my box with five dozens liquor jugs") ;
    assert(gs.length() == s.length()) ;
    assert(gs.size() == s.size()) ;
  }
  {
    string gs("93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000") ; //gold
    mstring s("93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000") ;
    assert(gs.length() == s.length()) ;
```

```
    assert(gs.size() == s.size()) ;
  }
  {
    string gs1("93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000") ; //gold
    string gs2(gs1) ;
    mstring s1("93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000") ;
    mstring s2(s1) ;
    assert(gs1.length() == s1.length()) ;
    assert(gs2.size() == s2.size()) ;

    assert_string_mstring_equal(gs1,s1) ;
    assert_string_mstring_equal(gs2,s2) ;
    assert_string_mstring_equal(s1,s2) ;
  }
  {
    string gs("UCSC EXTENSION",4) ; //gold
    mstring s("UCSC EXTENSION",4) ;
    cout << "gs = " << gs << endl ;
    cout << "s =  " << s << endl ;
    assert_string_mstring_equal(gs,s) ;
  }
  {
    string gs("UCSC EXTENSION",5,6) ; //gold
    mstring s("UCSC EXTENSION",5,6) ;
    cout << "gs = " << gs << endl ;
    cout << "s =  " << s << endl ;
    assert_string_mstring_equal(gs,s) ;
  }
}

/*-----------------------------------------------------------------
test element access
-----------------------------------------------------------------*/
void test_element_access() {
  {
    string gs("Return iterator to beginning (public member function )") ; //gold
    mstring s("abcu78") ;
    const char& gc = gs[3] ;
    const char& c =  s[3] ;
    cout << "gc = " << gc << endl ;
    assert(gc == c) ;
    const char& gc1 = gs[0] ;
    const char& c1 =  s[0] ;
    cout << "c1 = " << c1 << endl ;
    assert(gc1 != c1) ;
  }
}

/*-----------------------------------------------------------------
test_operator_plus_equal
-----------------------------------------------------------------*/
void test_operator_plus_equal() {
  {
    string gs1("Return") ; //gold
    string gs2(" gold") ; //gold
    gs1 += gs2 ;
    cout << "gs1 = " << gs1 << endl ;
    mstring s1("Return") ;
    mstring s2(" gold") ;
    s1 += s2 ;
    assert_string_mstring_equal(gs1,s1) ;
    assert_string_mstring_equal(gs2,s2) ;
```


VASUDEVAMURTHY

```
  }
  {
    string gs1("Return") ; //gold
    gs1 += " book" ;
    cout << "gs1 = " << gs1 << endl ;
    mstring s1("Return") ;
    s1 += " book" ;
    assert_string_mstring_equal(gs1,s1) ;
  }
  {
    string gs1("Return") ; //gold
    gs1 += 'Z' ;
    cout << "gs1 = " << gs1 << endl ;
    mstring s1("Return") ;
    s1 += 'Z' ;
    assert(gs1 == s1) ;
    s1 += 'Z' ;
    assert_string_mstring_not_equal(gs1,s1) ;
  }
}

/*-----------------------------------------------------------------
operator plus
-----------------------------------------------------------------*/
void test_operator_plus() {
  {
    string gs1("Return") ; //gold
    string gs2(" gold") ; //gold
    string gs3 = gs1 + gs2 ;
    cout << "gs3 = " << gs3 << endl ;

    mstring s1("Return") ;
    mstring s2(" gold") ;
    mstring s3 = s1 + s2 ;

    assert_string_mstring_equal(gs3,s3) ;
  }
  {
    string gs1("Return") ; //gold
    string gs3 = gs1 + "gold" ;
    cout << "gs3 = " << gs3 << endl ;

    mstring s1("Return") ;
    mstring s3 = s1 + "gold" ;

    assert_string_mstring_equal(gs3,s3) ;
  }
  {
    string gs1("Return") ; //gold
    string gs3 = "BOOK " + gs1  ;
    cout << "gs3 = " << gs3 << endl ;

    mstring s1("Return") ;
    mstring s3 = "BOOK " + s1  ;

    assert_string_mstring_equal(gs3,s3) ;
  }
#if 0
  {
    string gs3 = "BOOK " + " C++"  ; //gold
    cout << "gs3 = " << gs3 << endl ;

    mstring s3 = "BOOK " + " C++"  ;

    assert_string_mstring_equal(gs3,s3) ;
  }
```


VASUDEVAMURTHY

```
#endif
  {
    string gs1("Return") ; //gold
    string gs3 = 'P' + gs1  ;
    cout << "gs3 = " << gs3 << endl ;

    mstring s1("Return") ;
    mstring s3 = 'P' + s1  ;

    assert_string_mstring_equal(gs3,s3) ;
  }
  {
    string gs1("Return") ; //gold
    string gs3 = gs1 + 'P';
    cout << "gs3 = " << gs3 << endl ;

    mstring s1("Return") ;
    mstring s3 = s1 + 'P';

    assert_string_mstring_equal(gs3,s3) ;
  }
  {
    string gs1("Return") ; //gold
    string gs2 = " Hell" ;
    string gs3 = gs1 + gs2 + 'a' + gs2 + gs1 ;
    cout << "gs3 = " << gs3 << endl ;

    mstring s1("Return") ; //gold
    mstring s2 = " Hell" ;
    mstring s3 = s1 + s2 + 'a' + s2 + s1 ;
    cout << "s3 = " << s3 << endl ;

    assert_string_mstring_equal(gs3,s3) ;
  }
  {
    string gs1("abc") ;
    string gs2("xyz") ;
    string gs3("mnp") ;
    string gs4("zoo") ;
    string gs5 = gs1 + gs2 + "FOO1"+gs3 + gs4 + gs1 + gs2 + "FOO2"+gs3;
    cout << "gs5 = " << gs5 << endl ;

    string s1("abc") ;
    string s2("xyz") ;
    string s3("mnp") ;
    string s4("zoo") ;
    string s5 = s1 + s2 + "FOO1"+s3 + s4 + s1 + s2 + "FOO2"+s3;
    cout << "s5 = " << s5 << endl ;
    assert_string_mstring_equal(gs5,s5) ;
  }
}

/*-----------------------------------------------------------------
relational operators
-----------------------------------------------------------------*/
void test_equal_equal_operators() {
  {
    string gs1("Jag"); //gold
    string gs2("Jag"); //gold
    assert(gs1 == gs2) ;

    mstring s1("Jag") ;
    mstring s2("Jag") ;
    assert(s1 == s2) ;
  }
  {
```

VASUDEVAMURTHY

```
    string gs1("Jag"); //gold
    assert("Jag" == gs1) ;

    mstring s1("Jag") ;
    assert("Jag" == s1) ;
  }
  {
    string gs1("Jag"); //gold
    assert(gs1 == "Jag") ;

    mstring s1("Jag") ;
    assert(s1 == "Jag") ;
  }
}

/*-----------------------------------------------------------------
relational operators
-----------------------------------------------------------------*/
void test_not_equal_operators() {
  {
    string gs1("Jag"); //gold
    string gs2("jag"); //gold
    assert(gs1 != gs2) ;

    mstring s1("Jag") ;
    mstring s2("jag") ;
    assert(s1 != s2) ;
  }
  {
    string gs1("Jag"); //gold
    assert("jag" != gs1) ;

    mstring s1("Jag") ;
    assert("jag" != s1) ;
  }
  {
    string gs1("Jag"); //gold
    assert(gs1 != "jag") ;

    mstring s1("Jag") ;
    assert(s1 != "jag") ;
  }
}

/*-----------------------------------------------------------------
relational operators
-----------------------------------------------------------------*/
void test_less_than_operators() {
  {
    string gs1("Ja"); //gold
    string gs2("Jag"); //gold
    assert(gs1 < gs2) ;

    mstring s1("Ja") ;
    mstring s2("jag") ;
    assert(s1 < s2) ;
  }
  {
    string gs1("Jag"); //gold
    assert("Ja" < gs1) ;

    mstring s1("Jag") ;
    assert("Ja" < s1) ;
  }
  {
    string gs1("Jag"); //gold
```

```
    assert(gs1 < "ab") ;

    mstring s1("Jag") ;
    assert(s1 < "ab") ;
  }
}

/*-----------------------------------------------------------------
relational operators
-----------------------------------------------------------------*/
void test_less_than_equal_operators() {
  {
    string gs1("Ja"); //gold
    string gs2("Jag"); //gold
    assert(gs1 <= gs2) ;

    mstring s1("Ja") ;
    mstring s2("jag") ;
    assert(s1 <= s2) ;
  }
  {
    string gs1("Jag"); //gold
    assert("Ja" <= gs1) ;

    mstring s1("Jag") ;
    assert("Ja" <= s1) ;
  }
  {
    string gs1("Jag"); //gold
    assert(gs1 <= "ab") ;

    mstring s1("Jag") ;
    assert(s1 <= "ab") ;
  }
}

/*-----------------------------------------------------------------
relational operators
-----------------------------------------------------------------*/
void test_greater_than_operators() {
  {
    string gs1("XYfg"); //gold
    string gs2("Jag"); //gold
    assert(gs1 > gs2) ;

    mstring s1("XYfg") ;
    mstring s2("Jag") ;
    assert(s1 > s2) ;
  }
  {
    string gs1("Jag"); //gold
    assert("Ma" > gs1) ;

    mstring s1("Jag") ;
    assert("Ma" > s1) ;
  }
  {
    string gs1("Jag"); //gold
    assert(gs1 > "Aa") ;

    mstring s1("Jag") ;
    assert(s1 > "Aa") ;
  }
}

/*-----------------------------------------------------------------
```


VASUDEVAMURTHY

```
relational operators
----------------------------------------------------------------*/
void test_greater_than_equal_operators() {
  {
    string gs1("XYfg"); //gold
    string gs2("Jag"); //gold
    assert(gs1 >= gs2) ;

    mstring s1("XYfg") ;
    mstring s2("Jag") ;
    assert(s1 >= s2) ;
  }
  {
    string gs1("Jag"); //gold
    assert("Ma" >= gs1) ;

    mstring s1("Jag") ;
    assert("Ma" >= s1) ;
  }
  {
    string gs1("Jag"); //gold
    assert(gs1 >= "Aa") ;

    mstring s1("Jag") ;
    assert(s1 >= "Aa") ;
  }
}

/*----------------------------------------------------------------
relational operators
----------------------------------------------------------------*/
void test_swap() {
  {
    string gs1("XYfg"); //gold
    string gs2("Jag"); //gold

    cout << "gs1 = " << gs1 << endl ;
    cout << "gs2 = " << gs2 << endl ;

    swap(gs1,gs2) ;

    cout << "gs1 = " << gs1 << endl ;
    cout << "gs2 = " << gs2 << endl ;

    string s1("XYfg");
    string s2("Jag");
    swap(s1,s2) ;
    assert_string_mstring_equal(gs1,s1) ;
    assert_string_mstring_equal(gs2,s2) ;
  }
}


/*----------------------------------------------------------------
test bed
----------------------------------------------------------------*/
void testbed() {
  test_constructors() ;
  test_element_access() ;
  test_operator_plus_equal() ;
  test_operator_plus() ;
  test_equal_equal_operators();
  test_not_equal_operators();
  test_less_than_operators();
  test_less_than_equal_operators();
  test_greater_than_operators();
```


VASUDEVAMURTHY

```
  test_greater_than_equal_operators();
  test_swap() ;
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  testbed() ;
  return 0 ;
}

//EOF

```

VASUDEVAMURTHY

Chapter 8

Inheritance

8.1 Introduction

8.2 Why Inheritance?

VASUDEVAMURTHY

Why use inheritance?

```
class employee {
    char* name ;
    char* ssn ;
};

class part_time_employee: public employee {
     int how_many_hours;
     int pay_per_hour ;
};
```

1. Create a class (derived class) from an existing class (base class)
2. Do not reinvent the wheel
3. part_time_employee is-a employee. Create is-a relationship
4. Derived class is a specializaton of a base (parent) class
5. If a derived class is derived from one base, single inheritance
6. If a derived class is derived from two or more base, multiple inheritance

Figure 8.1: Need for inheritance

VASUDEVAMURTHY

8.3 Need for keyword: protected

Access previlages

```
class base {
private:
    int _x ;
public:
    int y ;
    int getx() const {return _x ; }

    base() {
        _x = 10, y = 20 ;
    }
};
```

```
base b ;
//b._x = 89 ; //Only class can access private
                members, not the instances
```

```
class derived: public base {
private:
    int _d ;
public:
    void f() {
        //_x = 100 ; //cannot access private member declared in class 'base'
        int x = getx() ; //OK can call public function
        y = 200 ; //OK. Can access public data
        _d = 6 ;
    }
};
```

```
derived d ;
d.y = 300 ; //OK. Can acess public data
//d._d = 10 ; //Only class can access
                private members, not the instances
```

Problem: 1) base class private members/data CANNOT be accessed by derived class
2) Only way to access is through public set/get functions or making EVERY PRIVATE members/data PUBLIC. Violation of DATA HIDING PRINCIPLE

Figure 8.2: Problem of accessing private data/methods by derived class

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: in1.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
base class
----------------------------------------------------------------*/
class base {
private:
    int _x ;
public:
    int y ;
    int getx() const {return _x ; }

    base() {
        _x = 10, y = 20 ;
    }
};

/*----------------------------------------------------------------
derived class
----------------------------------------------------------------*/
class derived: public base {
private:
    int _d ;
public:
    void f() {
        //_x = 100 ; //cannot access private member declared in class 'base'
        int x = getx() ; //OK can call public function
        y = 200 ; //OK. Can access public data
        _d = 6 ;
    }
};

/*----------------------------------------------------------------

----------------------------------------------------------------*/
int main() {
    base b ;
    //b._x = 89 ;// Only class can access private members, not the instances
    derived d ;
    d.y = 300 ; //OK. Can acess public data
    //d._d = 10 ; //Only class can access private members, not the instances
    return 0 ;
}

```

VASUDEVAMURTHY

Access previlages

```
class base {
private:
    int _x ;
protected:
    int _z;
public:
    int y ;
    int getx() const {return _x ; }
    base() {
        _x = 10, y = 20, _z = 87 ;
    }
};

class derived: public base {
private:
    int _d ;
public:
    void f() {
        //_x = 100 ; //cannot access private member declared in class 'base'
        int x = getx() ; //OK can call public function
        y = 200 ; //OK. Can access public data
        _z = 7 ; //OK. Can access protected base class data
        _d = 6 ;
    }
};
```

```
base b ;
//b._x = 89 ;// Only class can access private members,
                not the instances
//b._z = 5; //protected is same as private to
                ALL except derived class

derived d ;
d.y = 300 ; //OK. Can acess public data
//d._d = 10 ; //Only class can access
                private members, not the instances
//d._z = 5 ; //protected is same as private
                to ALL except derived class
```

Solution: 1) base class protected members/data CAN be accessed by derived class

2) Only derived class can access data/members of protected fields of base class. For all others classes and instances, protected is same as private

NO Violation of DATA HIDING PRINCIPLE

Figure 8.3: Protected same as public, only for the derived class

VASUDEVAMURTHY

Access previlages summary

```
class base {
private:
    int _x ;
protected:
    int _z;
public:
    int y ;
    int getx() const {return _x ; }
    base() {
        _x = 10, y = 20, _z = 87 ;
    }
};
```

```
class derived: public base {
private:
    int _d ;
public:
    void f() {
        //_x = 100 ;
        int x = getx() ;
        y = 200 ;
        _z = 7 ;
        _d = 6 ;
    }
};
```

1. base private: ONLY base class can access
   NO derived class from base class
   or instances of base class can access
   private members
2. base protected: ONLY base class and derived class
   can access private data/members
   protected base data/members are like
   public data/members for ONLY
   derived class.
   Instances of derived classes CANNOT access
   protected data/members. For all others, except,
   derived class, protected is same as private
3. base public: Any body can access any data/members

Figure 8.4: Summary

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: in2.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
base class
----------------------------------------------------------------*/
class base {
private:
    int _x ;
protected:
    int _z;
public:
    int y ;
    int getx() const {return _x ; }
    base() {
        _x = 10, y = 20, _z = 87 ;
    }
};

/*----------------------------------------------------------------
derived class
----------------------------------------------------------------*/
class derived: public base {
private:
    int _d ;
public:
    void f() {
        //_x = 100 ; //cannot access private member declared in class 'base'
        int x = getx() ; //OK can call public function
        y = 200 ; //OK. Can access public data
        _z = 7 ; //OK. Can access protected base class data
        _d = 6 ;
    }
};

/*----------------------------------------------------------------

----------------------------------------------------------------*/
int main() {
    base b ;
    //b._x = 89 ;// Only class can access private members, not the instances
    //b._z = 5; //protected is same as private to ALL except derived class
    derived d ;
    d.y = 300 ; //OK. Can acess public data
    //d._d = 10 ; //Only class can access private members, not the instances
    //d._z = 5 ; //protected is same as private to ALL except derived class
    return 0 ;
}

```

VASUDEVAMURTHY

8.4 Overriding or hiding base class member functions

VASUDEVAMURTHY

Overriding(hiding) base class member functions

```
class base {
public:
    void print() const { cout << "I am base print\n" ; }
    void print1() const {cout << "I am base print1\n" ; }
};

class derived: public base {
public:
    void print() const { cout << "I am derived print\n" ;}
};

int main() {
    base b ;
    derived d ;
    b.print() ;              I am base print
    d.print() ;              I am derived print
    d.base::print() ;        I am base print
    b.print1() ;             I am base print1
    d.print1() ;             I am base print1
    d.base::print1() ;       I am base print1
    base& ab = b ;
    derived& ad = d ;
    ab.print() ;             I am base print
    ad.print() ;             I am derived print
    ad.base::print() ;       I am base print
    ab.print1() ;            I am base print1
    ad.print1() ;            I am base print1
    ad.base::print1() ;      I am base print1
    base* bp = &(b) ;
    derived* dp = &(d) ;
    bp->print() ;            I am base print
    dp->print() ;            I am derived print
    dp->base::print() ;      I am base print
    bp->print1() ;           I am base print1
    dp->print1() ;           I am base print1
    dp->base::print1() ;     I am base print1
    return 0 ;
}
```

1. A member function declaration in a derived class hides/overrides all inherited functions with the same name.
2. d.base:: Using <base_name>:: can be used to bring hidden names of base class into derived class.

Figure 8.5: hiding and bring back base class member functions

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2010 Author: Jagadeesh Vasudevamurthy
Filename: in4.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
base class
----------------------------------------------------------------*/
class base {
public:
    void print() const { cout << "I am base print\n" ; }
    void print1() const {cout << "I am base print1\n" ; }
};

/*----------------------------------------------------------------
derived class
----------------------------------------------------------------*/
class derived: public base {
public:
    void print() const { cout << "I am derived print\n" ;}
};

/*----------------------------------------------------------------
I am base print
I am derived print
I am base print
I am base print1
I am base print1
I am base print1
I am base print
I am derived print
I am base print
I am base print1
I am base print1
I am base print1
I am base print
I am derived print
I am base print
I am base print1
I am base print1
I am base print1
----------------------------------------------------------------*/
int main() {
    base b ;
    derived d ;
    b.print() ;
    d.print() ;
    d.base::print() ;
    b.print1() ;
    d.print1() ;
    d.base::print1() ;
    base& ab = b ;
```


VASUDEVAMURTHY

```
    derived& ad = d ;
    ab.print() ;
    ad.print() ;
    ad.base::print() ;
    ab.print1() ;
    ad.print1() ;
    ad.base::print1() ;
    base* bp = &(b) ;
    derived* dp = &(d) ;
    bp->print() ;
    dp->print() ;
    dp->base::print() ;
    bp->print1() ;
    dp->print1() ;
    dp->base::print1() ;
    return 0 ;
}

```

VASUDEVAMURTHY

8.5 Writing constructor for a derived class

Writing constructor for base and derived class

```
class exam {
public:
    exam(const string& s, int m = 0, int f = 0):
        _sname(s),_midterm(m),_final(f) {
        cout << "exam constructor " << s << endl ;
    }
 private:
     string _sname ;
     int _midterm ;
     int _final ;
 };

class cs32exam : public exam {
public:
    cs32exam(const string& s,const string& e, int m = 0, int f = 0,int p = 0) :
      exam(s,m,f),_examname(e),_project(p) {
          cout << "cs32 exam constructor " << e << endl ;
      }
 private:
     string _examname ;
     int _project ;
 } ;
```

cs32exam p("sushma","cs32",65.75.85) ;

1.Raw memory allocated for derived class
2. Base class constructor is called first
3. Derived class data members are then initialized

NOTE THIS CALL: exam(s,m,f)

If you comment exam(s,m,f)
error C2512: 'exam' : no appropriate default constructor available

There can be many constructors for a class. That's why you need to call constructors on your own.

If the base class has only default constructor, example exam(), then you need NOT have to call exam(), explicitly in the derived class. See animal and dog examples, in future slides.

Figure 8.6: Writing constructor for a derived class

8.6 Writing destructor for a derived class

VASUDEVAMURTHY

Writing destructor for base and derived class

```
class exam {
public:
    ~exam() {
      cout << "exam destructor " << _sname << endl ;
    }
 }

class cs32exam : public exam {
public:
   ~cs32exam() {
      cout << "cs32exam destructor " << _examname << endl ;
      /* Code here to delete cs32exam data members only */
   }
}
```

When derived class destructor is called for an object base class destructor is called automatically. You need not have to call base class destructor explicitly, like calling base class constructor in the derived class. This is because there is only one destructor

```
 {
      cs32exam s("sushma","cs32",99,100,97) ;
 }
```

cs32exam destructor cs32
1. Kills derived class data members
2. Then kills base class data members using base class destructor(~exam)

cs32exam

KILL 2
KILL 1

Figure 8.7: Writing destructor for a derived class

VASUDEVAMURTHY

8.7 Writing copy constructor and equal operator for a derived class in a BAD way

Why does the implicit copy constructor calls the base class copy constructor and the defined copy constructor doesn't?

```
class B {
public:
  B() :_b(100) {
    cout << "B constructor " <<endl;
  }
  ~B() {
    cout << "B destructor " << endl;
  }
  B(const B& e) :_b(e._b){
    cout << "B copy constructor " << endl;
  }
  B& operator=(const B& e) {
    cout << "B = operator " << e._b << endl;
    if (&e != this) {
      _b = e._b;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const B& s) {
    o << s._b << " ";
    return o;
  }
  void setx(int x) {
    _b = x;
  }
private:
  int _b;
};
```

```
class D: public B {
public:
  D():_d(200) {
    cout << "D constructor " << endl;
  }
  ~D() {
    cout << "D destructor " << endl;
  }
  D(const D& e) :_d(e._d){
    cout << "D copy constructor " << endl;
  }
  D& operator=(const D& e) {
    cout << "D = operator " << e._d << endl;
    if (&e != this) {
      _d = e._d;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const D& s) {
    const B& b = s;
    o << b;
    o << s._d << " ";
    return o;
  }
private:
  int _d;
};
```

```
{
    cout << "-----1-----\n";
    D d1;
    cout << "-----2-----\n";
    D d2(d1);
    cout << "-----3-----\n";
    cout << "d1 = " << d1 << endl;
    cout << "d2 = " << d2 << endl;
    d1.setx(-89);
    cout << "d1 = " << d1 << endl;
    cout << "-----4-----\n";
    d2 = d1;
    cout << "-----5-----\n";
    cout << "d2 = " << d2 << endl;
}
```

```
-----1-----
B constructor
D constructor
-----2-----
B constructor (BUG or feature?)
D copy constructor
-----3-----
d1 = 100 200
d2 = 100 200
d1 = -89 200
-----4-----
D = operator 200 (DOES NOT CALL BASE =)
-----5-----
d2 = 100 200 (You want -89 200 ?)
D destructor
B destructor
D destructor
B destructor
```

Figure 8.8: Writing copy constructor and equal operator for a derived class, BAD way

8.8 Writing copy constructor and equal operator for a derived class in a CORRECT way

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
Filename: in8.cpp
----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

#ifdef _WIN32
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#endif

/*-------------------------------------------------------
class B
-------------------------------------------------------*/
class B {
public:
  B() :_b(100) {
    cout << "B constructor " <<endl;
  }
  ~B() {
    cout << "B destructor " << endl;
  }
  B(const B& e) :_b(e._b){
    cout << "B copy constructor " << endl;
  }
  B& operator=(const B& e) {
    cout << "B = operator " << e._b << endl;
    if (&e != this) {
      _b = e._b;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const B& s) {
    o << s._b << " ";
    return o;
  }
  void setx(int x) {
    _b = x;
  }
private:
  int _b;
};


/*-------------------------------------------------------
class D
-------------------------------------------------------*/
class D: public B {
public:
  D():_d(200) {
    cout << "D constructor " << endl;
  }
  ~D() {
    cout << "D destructor " << endl;
  }
  D(const D& e) :_d(e._d){
    cout << "D copy constructor " << endl;
  }
  D& operator=(const D& e) {
    cout << "D = operator " << e._d << endl;
    if (&e != this) {
      _d = e._d;
    }
```


VASUDEVAMURTHY

```
    return *this;
  }
  friend ostream& operator<<(ostream& o, const D& s) {
    const B& b = s;
    o << b;
    o << s._d << " ";
    return o;
  }
private:
  int _d;
};


/*-------------------------------------------------------
test bed
-------------------------------------------------------*/
int main() {
  {
    B b1;
    B b2(b1);
    b2 = b1;
    cout << "b2 = " << b2 << endl;
  }
  {
    cout << "-----1-----\n";
    D d1;
    cout << "-----2-----\n";
    D d2(d1);
    cout << "-----3-----\n";
    cout << "d1 = " << d1 << endl;
    cout << "d2 = " << d2 << endl;
    d1.setx(-89);
    cout << "d1 = " << d1 << endl;
    cout << "-----4-----\n";
    d2 = d1;
    cout << "-----5-----\n";
    cout << "d2 = " << d2 << endl;
  }
  return 0;
}

```

VASUDEVAMURTHY

Why does the implicit copy constructor calls the base class copy constructor and the defined copy constructor doesn't?

All base child constructors call the parent default constructor. This is how the standard is defined.
If you wanted the base class D to call B's copy constructor you have to explicitly ask for it

```
class B {
public:
  B() :_b(100) {
    cout << "B constructor " <<endl;
  }
  ~B() {
    cout << "B destructor " << endl;
  }
  B(const B& e) :_b(e._b){
    cout << "B copy constructor " << endl;
  }
  B& operator=(const B& e) {
    cout << "B = operator " << e._b << endl;
    if (&e != this) {
      _b = e._b;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const B& s) {
    o << s._b << " ";
    return o;
  }
  void setx(int x) {
    _b = x;
  }
private:
  int _b;
};
```

```
class D: public B {
public:
  D():_d(200) {
    cout << "D constructor " << endl;
  }
  ~D() {
    cout << "D destructor " << endl;
  }
  D(const D& e) :B(e)/*NOTE 1*/,_d(e._d){
    cout << "D copy constructor " << endl;
  }
  D& operator=(const D& e) {
    cout << "D = operator " << e._d << endl;
    if (&e != this) {
      B::operator=(e); //NOTE 2
      _d = e._d;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const D& s) {
    const B& b = s;
    o << b;
    o << s._d << " ";
    return o;
  }
private:
  int _d;
};
```

```
{
    cout << "-----1-----\n";
    D d1;
    cout << "-----2-----\n";
    D d2(d1);
    cout << "-----3-----\n";
    cout << "d1 = " << d1 << endl;
    cout << "d2 = " << d2 << endl;
    d1.setx(-89);
    cout << "d1 = " << d1 << endl;
    cout << "-----4-----\n";
    d2 = d1;
    cout << "-----5-----\n";
    cout << "d2 = " << d2 << endl;
}
```

```
-----1-----
B constructor
D constructor
-----2-----
B copy constructor (CALLS BASE COPY)
D copy constructor
-----3-----
d1 = 100 200
d2 = 100 200
d1 = -89 200
-----4-----
D = operator 200
B = operator -89(CALLS BASE = operator)
-----5-----
d2 = -89 200 (CORRECT)
```

Figure 8.9: Writing copy constructor and equal operator for a derived class, CORRECT way

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2014 Author: Jagadeesh Vasudevamurthy
Filename: in9.cpp
----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

#ifdef _WIN32
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#endif

/*-------------------------------------------------------
class B
-------------------------------------------------------*/
class B {
public:
  B() :_b(100) {
    cout << "B constructor " <<endl;
  }
  ~B() {
    cout << "B destructor " << endl;
  }
  B(const B& e) :_b(e._b){
    cout << "B copy constructor " << endl;
  }
  B& operator=(const B& e) {
    cout << "B = operator " << e._b << endl;
    if (&e != this) {
      _b = e._b;
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const B& s) {
    o << s._b << " ";
    return o;
  }
  void setx(int x) {
    _b = x;
  }
private:
  int _b;
};


/*-------------------------------------------------------
class D
-------------------------------------------------------*/
class D: public B {
public:
  D():_d(200) {
    cout << "D constructor " << endl;
  }
  ~D() {
    cout << "D destructor " << endl;
  }
  D(const D& e) :B(e)/*NOTE THIS CALL*/,_d(e._d){
    cout << "D copy constructor " << endl;
  }
  D& operator=(const D& e) {
    cout << "D = operator " << e._d << endl;
    if (&e != this) {
      B::operator=(e); //Note this call.base of e (RHS) is copied to base of this object(on LHS)
      _d = e._d;
```


VASUDEVAMURTHY

```
    }
    return *this;
  }
  friend ostream& operator<<(ostream& o, const D& s) {
    const B& b = s;
    o << b;
    o << s._d << " ";
    return o;
  }
private:
  int _d;
};


/*-------------------------------------------------------
test bed
-------------------------------------------------------*/
int main() {
  {
    B b1;
    B b2(b1);
    b2 = b1;
    cout << "b2 = " << b2 << endl;
  }
  {
    cout << "-----1-----\n";
    D d1;
    cout << "-----2-----\n";
    D d2(d1);
    cout << "-----3-----\n";
    cout << "d1 = " << d1 << endl;
    cout << "d2 = " << d2 << endl;
    d1.setx(-89);
    cout << "d1 = " << d1 << endl;
    cout << "-----4-----\n";
    d2 = d1;
    cout << "-----5-----\n";
    cout << "d2 = " << d2 << endl;
  }
  return 0;
}

```

VASUDEVAMURTHY

8.9 Writing copy constructor for a derived class

Writing copy constructor for base and derived class

```
class exam {
public:
  exam(const exam& e):
     _sname(e._sname), _midterm(e._midterm),_final(e._final) {
          cout << "exam copy constructor " << e._sname << endl ;
      }
 private:
     string _sname ;
     int _midterm ;
     int _final ;
 };

class cs32exam : public exam {
public:
   cs32exam(const cs32exam& e):
        exam(e),_examname(e._examname),_project(e._project)  {
             cout << "cs32exam copy constructor " << e._examname << endl ;
         }
 private:
     string _examname ;
     int _project ;
 } ;
```

NOTE THIS CALL: exam(e)

If you comment exam(e)
error C2512: 'exam' : no appropriate default constructor available

Figure 8.10: Writing copy constructor for a derived class

8.10 Writing equal operator for a derived class

VASUDEVAMURTHY

Writing equal operator for base and derived class

```
class exam {
public:
      exam& operator=(const exam& e) {
              cout << "exam = operator " << e._sname << endl;
              if (&e != this) {
                  _sname = e._sname ;
                  _midterm = e._midterm ;
                  _final = e._final ;
              }
              return *this ;
          }
 }

class cs32exam : public exam {
public:
cs32exam& operator=(const cs32exam& e) {
    if (&e != this) {
     cout << "cs32exam = operator " << e._examname << endl ;
     exam::operator=(e) ; //Note this call
                //base of e (RHS) is copied to base of this object(on LHS)
     _examname = e._examname ;
     _project = e._project;
    }
    return *this ;
}
```

Without exam::operator=
cs32exam operator= will be called.
INFINITE RECURSION

NOTE THIS CALL: exam::operator=(e)

if you comment operator=(e), base of e will NOT be copied to base of left
Only derived portion of e will be copied to the left derived portion

Figure 8.11: Writing equal operator for a derived class

VASUDEVAMURTHY

8.11 Writing insertion operator for a derived class

Figure 8.12: Writing insertion operator for a derived class

8.12 Using base and derived class objects

VASUDEVAMURTHY

1. Using base and derived class

```
void test1() {
    exam s("sushma",52,80) ;
    cout << s << endl ;

    exam s1(s) ;
    cout << s1 << endl ;

    exam s2("neema",89,78) ;
    cout << s2 << endl ;

    s1 = s2 ;
    cout << s1 << endl ;
    s1.who_am_i() ;
}
```

```
exam constructor sushma
sushma  52 80

exam copy constructor sushma
sushma  52 80

exam constructor neema
neema  89 78

exam = operator neema
neema  89 78
I am exam class neema

exam destructor neema (for s2)
exam destructor neema (for s1)
exam destructor sushma (for s)
```

Figure 8.13: Using only base

VASUDEVAMURTHY

2. Using base and derived class

```
void test2() {
   cs32exam s("sushma","cs32",99,100,97) ;
   cout << s << endl ;

   cs32exam s1(s) ;
   cout << s1 << endl ;

   cs32exam s2("Bill","cs32",56,77,88) ;
   cout << s2 << endl ;

   s1 = s2 ;
   cout << s1 << endl ;

   s1.who_am_i() ;
   s1.exam::who_am_i() ;
}
```

```
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 cs32  97

exam copy constructor sushma
cs32exam copy constructor cs32
sushma  99 100 cs32  97

exam constructor Bill
cs32 exam constructor cs32
Bill  56 77 cs32  88

cs32exam = operator cs32
exam = operator Bill
Bill  56 77 cs32  88

I am cs32exam class cs32
I am exam class Bill

cs32exam destructor cs32 (s2)
exam destructor Bill (s2)
cs32exam destructor cs32 (s1)
exam destructor Bill (s1)
cs32exam destructor cs32 (s)
exam destructor sushma (s)
```

Figure 8.14: Using only derived

VASUDEVAMURTHY

3. Using base and derived class

```
void test3() {
    exam* b = new exam("sushma",52,80) ;
    cout << *b << endl ;


    cs32exam* d = new cs32exam
            ("sushma","cs32",99,100,97) ;
    cout << *d << endl ;

    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;

    delete b ;
    delete d ;
}
```

```
exam constructor sushma
sushma  52 80

exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 cs32  97

I am exam class sushma
I am cs32exam class cs32
I am exam class sushma
exam destructor sushma

cs32exam destructor cs32
exam destructor sushma
```

Figure 8.15: Using derived objects on heap

4. Using base and derived class

```
void test4() {
    exam* b = new exam("sushma",52,80) ;
    cout << *b << endl ;

    exam* d = new cs32exam
          ("sushma","cs32",99,100,97) ;

    cout << *d << endl ; /* static binding */

    b->who_am_i() ;
    d->who_am_i() ;

    d->exam::who_am_i() ;

    delete b ;
    delete d ;
}
```

```
exam constructor sushma
sushma  52 80

exam constructor sushma
cs32 exam constructor cs32

sushma  99 100
   (why it did not print cs32 and 97?)

I am exam class sushma
I am exam class sushma
    (why it did not print cs32exam? )

I am exam class sushma

exam destructor sushma
exam destructor sushma
   (Why destructor of cs32exam is not called)
```

Problem of static binding. Pointer d is thought of
base exam class, although it is pointing to derived cs32exam class

Figure 8.16: Problems of static binding

VASUDEVAMURTHY

5. Using base and derived class

```
void test5() {
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;

    exam* b = &bi ;
    cout << *b << endl ;
    exam* d = &di ; /* static binding */
    cout << *d << endl ;

    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;
}
```

```
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32

sushma  52 80

sushma  99 100

I am exam class sushma
I am exam class sushma
I am exam class sushma

cs32exam destructor cs32(di)
exam destructor sushma(di)
exam destructor sushma(bi)
```

Problem of static binding. Pointer d is thought of base exam class, although it is pointing to derived cs32exam class

Note that: when di went out of scope both cs32exam and exam destructor is called.

Figure 8.17: Problems of static binding

VASUDEVAMURTHY

6. Using base and derived class

```
void test6() {
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;

    exam& b = bi ;
    cout << b << endl ;
    exam& d = di ; /* static binding */
    cout << d << endl ;

    b.who_am_i() ;
    d.who_am_i() ;
    d.exam::who_am_i() ;
}
```

```
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32

sushma  52 80

sushma  99 100

I am exam class sushma
I am exam class sushma
I am exam class sushma

cs32exam destructor cs32(di)
exam destructor sushma(di)
exam destructor sushma(bi)
```

Problem of static binding. Alias object d is thought of base exam class, although it is an object of the derived cs32exam class

Note that: when di went out of scope both cs32exam and exam destructor is called.

Figure 8.18: Problems of static binding

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: in7.cpp
-----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
class exam
-------------------------------------------------------*/
class exam {
public:
   exam(const string& s, int m = 0, int f = 0):_sname(s),_midterm(m),_final(f) {
      cout << "exam constructor " << s << endl ;
   }
   ~exam() {
     cout << "exam destructor " << _sname << endl ;
   }
   exam(const exam& e):_sname(e._sname), _midterm(e._midterm),_final(e._final) {
      cout << "exam copy constructor " << e._sname << endl ;
   }
   exam& operator=(const exam& e) {
      cout << "exam = operator " << e._sname << endl;
      if (&e != this) {
         _sname = e._sname ;
         _midterm = e._midterm ;
         _final = e._final ;
      }
      return *this ;
   }
   friend ostream& operator<<(ostream& o, const exam& s) {
      o << s._sname << "  " << s._midterm << " " << s._final << " " ;
      return o ;
   }
   void who_am_i() const {
      cout << "I am exam class " << _sname << endl ;
   }
private:
   string _sname ;
   int _midterm ;
   int _final ;
};

/*-------------------------------------------------------
cs32 exam
-------------------------------------------------------*/
class cs32exam : public exam {
public:
   cs32exam(const string& s,const string& e, int m = 0, int f = 0,int p = 0) :
     exam(s,m,f)/*Note this call*/,_examname(e),_project(p) {
         cout << "cs32 exam constructor " << e << endl ;
     }
```


VASUDEVAMURTHY

```
56      ~cs32exam() {
57        cout << "cs32exam destructor " << _examname << endl ;
58      }
59      cs32exam(const cs32exam& e):exam(e) /*NOTE THIS CALL */,_examname(e._examname),_project(e._project)  {
60         cout << "cs32exam copy constructor " << e._examname << endl ;
61      }
62      cs32exam& operator=(const cs32exam& e) {
63         if (&e != this) {
64            cout << "cs32exam = operator " << e._examname << endl ;
65            exam::operator=(e) ; //Note this call.  base of e (RHS) is copied to base of this object(on LHS)
66            _examname = e._examname ;
67            _project = e._project;
68         }
69         return *this ;
70      }
71      friend ostream& operator<<(ostream& o, const cs32exam& s) {
72         const exam& e = s ;
73         o << e ;
74         o << s._examname<< "  " << s._project<< " ";
75         return o ;
76      }
77      void who_am_i() const {
78         cout << "I am cs32exam class " << _examname << endl ;
79      }
80 private:
81    string _examname ;
82    int _project ;
83 } ;
84
85
86 /*-------------------------------------------------------
87 -------------test1------------
88 exam constructor sushma
89 sushma  52 80
90 exam copy constructor sushma
91 sushma  52 80
92 exam constructor neema
93 neema  89 78
94 exam = operator neema
95 neema  89 78
96 I am exam class neema
97 exam destructor neema
98 exam destructor neema
99 exam destructor sushma
100 -------------------------------------------------------*/
101 void test1() {
102    cout << "-------------test1------------\n" ;
103    exam s("sushma",52,80) ;
104    cout << s << endl ;
105    exam s1(s) ;
106    cout << s1 << endl ;
107    exam s2("neema",89,78) ;
108    cout << s2 << endl ;
109    s1 = s2 ;
110    cout << s1 << endl ;
```

```
   s1.who_am_i() ;
}

/*-------------------------------------------------------
-------------test2------------
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 cs32  97
exam copy constructor sushma
cs32exam copy constructor cs32
sushma  99 100 cs32  97
exam constructor Bill
cs32 exam constructor cs32
Bill  56 77 cs32  88
cs32exam = operator cs32
exam = operator Bill
Bill  56 77 cs32  88
I am cs32exam class cs32
I am exam class Bill
cs32exam destructor cs32
exam destructor Bill
cs32exam destructor cs32
exam destructor Bill
cs32exam destructor cs32
exam destructor sushma
-------------------------------------------------------*/
void test2() {
   cout << "-------------test2------------\n" ;
   cs32exam s("sushma","cs32",99,100,97) ;
   cout << s << endl ;;
   cs32exam s1(s) ;
   cout << s1 << endl ;
   cs32exam s2("Bill","cs32",56,77,88) ;
   cout << s2 << endl ; ;
   s1 = s2 ;
   cout << s1 << endl ;;
   s1.who_am_i() ;
   s1.exam::who_am_i() ;
}

/*-------------------------------------------------------
-------------test3------------
exam constructor sushma
sushma  52 80
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 cs32  97
I am exam class sushma
I am cs32exam class cs32
I am exam class sushma
exam destructor sushma
cs32exam destructor cs32
exam destructor sushma
-------------------------------------------------------*/
void test3() {
```


VAIJDEVAMURTHY

```
166     cout << "-------------test3------------\n" ;
167     exam* b = new exam("sushma",52,80) ;
168     cout << *b << endl ;
169     cs32exam* d = new cs32exam("sushma","cs32",99,100,97) ;
170     cout << *d << endl ;
171     b->who_am_i() ;
172     d->who_am_i() ;
173     d->exam::who_am_i() ;
174     delete b ;
175     delete d ;
176 }
177
178 /*-------------------------------------------------------
179 -------------test4------------
180 exam constructor sushma
181 sushma  52 80
182 exam constructor sushma
183 cs32 exam constructor cs32
184 sushma  99 100 <-----------------WHY
185 I am exam class sushma
186 I am exam class sushma <-------- WHY
187 I am exam class sushma
188 exam destructor sushma
189 exam destructor sushma <---------WHY
190 -------------------------------------------------------*/
191 void test4() {
192     cout << "-------------test4------------\n" ;
193     exam* b = new exam("sushma",52,80) ;
194     cout << *b << endl ;
195     exam* d = new cs32exam("sushma","cs32",99,100,97) ;
196     cout << *d << endl ; /* static binding */
197     b->who_am_i() ;
198     d->who_am_i() ;
199     d->exam::who_am_i() ;
200     delete b ;
201     delete d ;
202 }
203
204 /*-------------------------------------------------------
205 -------------test5------------
206 exam constructor sushma
207 exam constructor sushma
208 cs32 exam constructor cs32
209 sushma  52 80
210 sushma  99 100 <-----------WHY
211 I am exam class sushma
212 I am exam class sushma <--------------WHY
213 I am exam class sushma
214 cs32exam destructor cs32
215 exam destructor sushma
216 exam destructor sushma
217 -------------------------------------------------------*/
218 void test5() {
219     cout << "-------------test5------------\n" ;
220     exam bi("sushma",52,80) ;
```


VAIDEVAMURTHY

```
    cs32exam di("sushma","cs32",99,100,97) ;
    exam* b = &bi ;
    cout << *b << endl ;
    exam* d = &di ; /* static binding */
    cout << *d << endl ;
    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;
}

/*-------------------------------------------------------
-------------test6------------
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 <-----------WHY
I am exam class sushma
I am exam class sushma <--------------WHY
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma
-------------------------------------------------------*/
void test6() {
    cout << "-------------test6------------\n" ;
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;
    exam& b = bi ;
    cout << b << endl ;
    exam& d = di ; /* static binding */
    cout << d << endl ;
    b.who_am_i() ;
    d.who_am_i() ;
    d.exam::who_am_i() ;
}

/*-------------------------------------------------------
main
-------------------------------------------------------*/
int main() {
    test1() ;
    test2() ;
    test3() ;
    test4() ;
    test5() ;
    test6() ;
    return 0 ;
}
```

VAMUDEVAMURTHY

8.13 Problems of static binding and need for polymorphism

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Problem of static binding and need for Polymorphism

```
class animal {
public:
    animal() {cout << "animal constructor\n";}
    void who_am_i() { cout << "I am an animal\n" ; }
    ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
    dog() {cout << "dog constructor\n";}
    void who_am_i() { cout << "I am a dog\n" ; }
    ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
    cat() {cout << "cat constructor\n";}
    void who_am_i() { cout << "I am a cat\n" ; }
    ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
    lion() {cout << "lion constructor\n";}
    void who_am_i() { cout << "I am lion\n" ; }
    ~lion() {cout << "lion destructor\n";}
};
```

```
void object_without_polymorphism() {
    dog d ;
    cat c ;
    lion n ;
    cat c1 ;

    animal* a[4] ;
    a[0] = &d ;
    a[1] = &c ;
    a[2] = &n ;
    a[3] = &c1 ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
}
```

```
animal constructor
dog constructor
animal constructor
cat constructor
animal constructor
lion constructor
animal constructor
cat constructor

I am an animal
I am an animal
I am an animal
I am an animal
cat destructor
animal destructor
lion destructor
animal destructor
cat destructor
animal destructor
dog destructor
animal destructor
```

Static binding. Every object is thought as animal

Correct because objects are destroyed

Figure 8.19: Problems of static binding with objects

VASUDEVAMURTHY

Problem of static binding and need for Polymorphism

```
class animal {
public:
    animal() {cout << "animal constructor\n";}
    void who_am_i() { cout << "I am an animal\n" ; }
    ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
    dog() {cout << "dog constructor\n";}
    void who_am_i() { cout << "I am a dog\n" ; }
    ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
    cat() {cout << "cat constructor\n";}
    void who_am_i() { cout << "I am a cat\n" ; }
    ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
    lion() {cout << "lion constructor\n";}
    void who_am_i() { cout << "I am lion\n" ;}
    ~lion() {cout << "lion destructor\n";}
};
```

```
void pointer_without_polymorphism() {
    animal* a[4] ;
    a[0] = new dog() ;
    a[1] = new cat() ;
    a[2] = new lion() ;
    a[3] = new cat() ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
    for (int i = 0; i < 4; i++) {
        delete a[i] ;
    }
}
```

```
animal constructor
dog constructor
animal constructor
cat constructor
animal constructor
lion constructor
animal constructor
cat constructor
```

```
I am an animal
I am an animal
I am an animal
I am an animal
```

Static binding. Every object is thought as animal

```
animal destructor
animal destructor
animal destructor
animal destructor
```

NOT Correct because objects are thought as animals

Figure 8.20: Problems of static binding with pointers to objects

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: need_for_poly.cpp
-----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
            Polymorphism
-------------------------------------------------------*/
class animal {
public:
   animal() {cout << "animal constructor\n";}
   void who_am_i() { cout << "I am an animal\n" ; }
   ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
   dog() {cout << "dog constructor\n";}
   void who_am_i() { cout << "I am a dog\n" ; }
   ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
   cat() {cout << "cat constructor\n";}
   void who_am_i() { cout << "I am a cat\n" ; }
   ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
   lion() {cout << "lion constructor\n";}
   void who_am_i() { cout << "I am lion\n" ; }
   ~lion() {cout << "lion destructor\n";}
};

/*-------------------------------------------------------
animal constructor
dog constructor
animal constructor
cat constructor
animal constructor
lion constructor
animal constructor
cat constructor
I am an animal
I am an animal
I am an animal
I am an animal
cat destructor
```

VASUDEVAMURTHY

```
animal destructor
lion destructor
animal destructor
cat destructor
animal destructor
dog destructor
animal destructor
-----------------------------------------------------------*/
void object_without_polymorphism() {
    dog d ;
    cat c ;
    lion n ;
    cat c1 ;
    animal* a[4] ;
    a[0] = &d ;
    a[1] = &c ;
    a[2] = &n ;
    a[3] = &c1 ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
}

/*-----------------------------------------------------------
 animal constructor
dog constructor
animal constructor
cat constructor
animal constructor
lion constructor
animal constructor
cat constructor
I am an animal
I am an animal
I am an animal
I am an animal
animal destructor
animal destructor
animal destructor
animal destructor
-----------------------------------------------------------*/
void pointer_without_polymorphism() {
    animal* a[4] ;
    a[0] = new dog() ;
    a[1] = new cat() ;
    a[2] = new lion() ;
    a[3] = new cat() ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
    for (int i = 0; i < 4; i++) {
        delete a[i] ;
    }
}

```

VAJDEVAMURTHY

```
/*-------------------------------------------------------
        main
-------------------------------------------------------*/
void main() {
   object_without_polymorphism() ;
   pointer_without_polymorphism() ;
}


```

VASUDEVAMURTHY

8.14 Polymorphism through keyword virtual

Dynamic binding and Polymorphism

```
class animal {
public:
    animal() {cout << "animal constructor\n";}
    virtual void who_am_i() { cout << "I am an animal\n" ; }
    virtual ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
    dog() {cout << "dog constructor\n";}
    void who_am_i() { cout << "I am a dog\n" ; }
    ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
    cat() {cout << "cat constructor\n";}
    void who_am_i() { cout << "I am a cat\n" ; }
    ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
    lion() {cout << "lion constructor\n";}
    void who_am_i() { cout << "I am lion\n" ; }
    ~lion() {cout << "lion destructor\n";}
};
```

```
void object_polymorphism() {
    dog d ;

    cat c ;

    lion n ;

    cat c1 ;

    animal* a[4] ;
    a[0] = &d ;
    a[1] = &c ;
    a[2] = &n ;
    a[3] = &c1 ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
}
```

```
animal constructor
dog constructor
animal constructor
cat constructor
animal constructor
lion constructor
animal constructor
cat constructor

I am a dog
I am a cat
I am lion
I am a cat

cat destructor
animal destructor
lion destructor
animal destructor
cat destructor
animal destructor
dog destructor
animal destructor
```

Dynamic binding. Every object is NOT thought as just animal. It exactly knows who it is

Figure 8.21: Dynamic binding of objects

VASUDEVAMURTHY

Dynamic binding and Polymorphism

```
class animal {
public:
    animal() {cout << "animal constructor\n";}
    virtual void who_am_i() { cout << "I am an animal\n" ; }
    virtual ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
    dog() {cout << "dog constructor\n";}
    void who_am_i() { cout << "I am a dog\n" ; }
    ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
    cat() {cout << "cat constructor\n";}
    void who_am_i() { cout << "I am a cat\n" ; }
    ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
    lion() {cout << "lion constructor\n";}
    void who_am_i() { cout << "I am lion\n" ; }
    ~lion() {cout << "lion destructor\n";}
};
```

```
void pointer_polymorphism() {
    animal* a[4] ;
    a[0] = new dog() ;
    a[1] = new cat() ;
    a[2] = new lion() ;
    a[3] = new cat() ;

    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }

    for (int i = 0; i < 4; i++) {
        delete a[i] ;
    }
}
```

```
animal constructor
dog constructor
animal constructor
cat constructor
animal constructor
lion constructor
animal constructor
cat constructor
I am a dog
I am a cat
I am lion
I am a cat

dog destructor
animal destructor
cat destructor
animal destructor
lion destructor
animal destructor
cat destructor
animal destructor
```

Dynamic binding. Every object is NOT thought as just animal. It exactly knows who it is

Note that both base and derived destructors are called because base destructor is virtual
virtual ~animal() {cout << "animal destructor\n";}

DREAM animal* a[4] which stores dog, cats and lion
Hetrogenous container

Figure 8.22: Dynamic binding of pointers to objects

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2009 Author: Jagadeesh Vasudevamurthy
Filename: poly.cpp
-----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
            Polymorphism
-------------------------------------------------------*/
class animal {
public:
   animal() {cout << "animal constructor\n";}
   virtual void who_am_i() { cout << "I am an animal\n" ; }
   virtual ~animal() {cout << "animal destructor\n";}
};

class dog:public animal {
public:
   dog() {cout << "dog constructor\n";}
   void who_am_i() { cout << "I am a dog\n" ; }
   ~dog() {cout << "dog destructor\n";}
};

class cat:public animal {
public:
   cat() {cout << "cat constructor\n";}
   void who_am_i() { cout << "I am a cat\n" ; }
   ~cat() {cout << "cat destructor\n";}
};

class lion:public animal {
public:
   lion() {cout << "lion constructor\n";}
   void who_am_i() { cout << "I am lion\n" ; }
   ~lion() {cout << "lion destructor\n";}
};

/*-------------------------------------------------------

-------------------------------------------------------*/
void object_polymorphism() {
   dog d ;
   cat c ;
   lion n ;
   cat c1 ;
   animal* a[4] ;
   a[0] = &d ;
   a[1] = &c ;
   a[2] = &n ;
   a[3] = &c1 ;
   for (int i = 0; i < 4; i++) {
```


VASUDEVAMURTHY

```
        a[i]->who_am_i() ;
    }
}

/*-------------------------------------------------------

-------------------------------------------------------*/
void pointer_polymorphism() {
    animal* a[4] ;
    a[0] = new dog() ;
    a[1] = new cat() ;
    a[2] = new lion() ;
    a[3] = new cat() ;
    for (int i = 0; i < 4; i++) {
        a[i]->who_am_i() ;
    }
    for (int i = 0; i < 4; i++) {
        delete a[i] ;
    }
}

/*-------------------------------------------------------
            main
-------------------------------------------------------*/
void main() {
    object_polymorphism() ;
    pointer_polymorphism() ;
}

```

VASUDEVAMURTHY

8.15 Rewriting exam program using polymorphism

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

4. Problem of static binding corrected

```
1. virtual ~exam() { }
2. virtual ostream& print(ostream& o) const {}
3. virtual void who_am_i() const {}
4.virtual ostream& print(ostream& o) const {}
```

```
void test4() {
    exam* b = new exam("sushma",52,80) ;
    cout << *b << endl ;
    exam* d = new cs32exam("sushma","cs32",99,100,97)
    cout << *d << endl ; /* Dynamic binding */
    d->print(cout) ; cout << endl ; /* dynamic binding */
    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;
    delete b ;
    delete d ;
}
```

Dynamic binding

Dynamic binding

Dynamic binding

```
exam constructor sushma
sushma  52 80
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 cs32  97
sushma  99 100 cs32  97
I am exam class sushma
I am cs32exam class cs32
I am exam class sushma
exam destructor sushma
cs32exam destructor cs32
exam destructor sushma
```

```
friend ostream& operator<<(ostream& o, const exam& s) {
        s.print(o) ;
        return o ;
    }
    virtual ostream& print(ostream& o) const {
        o << _sname << "  " << _midterm << " " << _final << " " ;
        return o ;
    }
```

exam class

Because of virtual

```
friend ostream& operator<<(ostream& o, const cs32exam& s) {
        s.print(o) ;
    }
    virtual ostream& print(ostream& o) const {
        this->exam::print(o) ;
        o << _examname<< "  " << _project<< " ";
        return o ;
    }
```

virtual

cs32exam class

Figure 8.23: Problems of static binding fixed

VASUDEVAMURTHY

5. Problem of static binding corrected

```
1. virtual ~exam() { }
2. virtual ostream& print(ostream& o) const {}
3. virtual void who_am_i() const {}
4.virtual ostream& print(ostream& o) const {
        o << *this ;
        return o ;
 }

void test5() {
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;
    exam* b = &bi ;
    cout << *b << endl ;
    exam* d = &di ; /* Dynamic binding */
    cout << *d << endl ;
    d->print(cout) ; cout << endl ;
    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;
}
```

Dynamic binding

```
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 cs32 97
sushma  99 100 cs32  97
I am exam class sushma
I am cs32exam class cs32
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma
```

Figure 8.24: Problems of static binding fixed

VASUDEVAMURTHY

6. Problem of static binding corrected

```
1. virtual ~exam() { }
2. virtual ostream& print(ostream& o) const {}
3. virtual void who_am_i() const {}
4.virtual ostream& print(ostream& o) const {
        o << *this ;
        return o ;
 }

void test6() {
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;
    exam& b = bi ;
    cout << b << endl ;
    exam& d = di ; /* dynamic binding */
    cout << d << endl ;
    d.print(cout) ; cout << endl ;
    b.who_am_i() ;
    d.who_am_i() ;
    d.exam::who_am_i() ;
}
```

Dynamic binding

```
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 cs32 97
sushma  99 100 cs32  97
I am exam class sushma
I am cs32exam class cs32
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma
```

Figure 8.25: Problems of static binding fixed

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: in7_corrected.cpp
----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
class exam
-------------------------------------------------------*/
class exam {
public:
   exam(const string& s, int m = 0, int f = 0):_sname(s),_midterm(m),_final(f) {
      cout << "exam constructor " << s << endl ;
   }
   virtual ~exam() { cout << "exam destructor " << _sname << endl ; }
   exam(const exam& e):_sname(e._sname), _midterm(e._midterm),_final(e._final) {
      cout << "exam copy constructor " << e._sname << endl ;
   }
   exam& operator=(const exam& e) {
      cout << "exam = operator " << e._sname << endl;
      if (&e != this) {
         _sname = e._sname ;
         _midterm = e._midterm ;
         _final = e._final ;
      }
      return *this ;
   }
   friend ostream& operator<<(ostream& o, const exam& s) {
      s.print(o) ;
      return o ;
   }
   virtual ostream& print(ostream& o) const {
      o << _sname << "  " << _midterm << " " << _final << " " ;
      return o ;
   }
   virtual void who_am_i() const {
      cout << "I am exam class " << _sname << endl ;
   }
   void pass_by_reference(exam*& e) {
     e->who_am_i() ;
   }
   void pass_by_value(exam* e) {
     e->who_am_i() ;
   }
private:
   string _sname ;
   int _midterm ;
   int _final ;
};

/*-------------------------------------------------------
```


VASUDEVAMURTHY

```
cs32 exam
-------------------------------------------------------*/
class cs32exam : public exam {
public:
   cs32exam(const string& s,const string& e, int m = 0, int f = 0,int p = 0) :
    exam(s,m,f)/*Note this call*/,_examname(e),_project(p) {
        cout << "cs32 exam constructor " << e << endl ;
    }
    ~cs32exam() { cout << "cs32exam destructor " << _examname << endl ; }
    cs32exam(const cs32exam& e):exam(e)/* NOTE */,_examname(e._examname),_project(e._project)  {
        cout << "cs32exam copy constructor " << e._examname << endl ;
    }
    cs32exam& operator=(const cs32exam& e) {
        if (&e != this) {
            cout << "cs32exam = operator " << e._examname << endl ;
            exam::operator=(e) ; //Note this call.  base of e is copied base of this e
            _examname = e._examname ;
            _project = e._project;
        }
        return *this ;
    }
    friend ostream& operator<<(ostream& o, const cs32exam& s) {
        s.print(o) ;
    }
    virtual ostream& print(ostream& o) const {
        this->exam::print(o) ;
        o << _examname<< "  " << _project<< " ";
        return o ;
    }
    void who_am_i() const {
        cout << "I am cs32exam class " << _examname << endl ;
    }
private:
   string _examname ;
   int _project ;
} ;

/*-------------------------------------------------------
-------------test4 OLD------------
exam constructor sushma
sushma  52 80
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 <-----------------WHY
I am exam class sushma
I am exam class sushma <-------- WHY
I am exam class sushma
exam destructor sushma
exam destructor sushma <---------WHY

-------------test4 NOW ------------
exam constructor sushma
sushma  52 80
exam constructor sushma
cs32 exam constructor cs32
```


VASUDEVAMURTHY

```
sushma  99 100 cs32  97 <----  SOLVED
sushma  99 100 cs32  97 <----- SOLVED
I am exam class sushma
I am cs32exam class cs32 <--------- SOLVED
I am exam class sushma
exam destructor sushma
cs32exam destructor cs32 <----------SOLVED
exam destructor sushma
-----------------------------------------------------------*/
void test4() {
    cout << "-------------test4------------\n" ;
    exam* b = new exam("sushma",52,80) ;
    cout << *b << endl ;
    exam* d = new cs32exam("sushma","cs32",99,100,97) ;
    cout << *d << endl ; /* static binding */
    d->print(cout) ; cout << endl ; /* dynamic binding */
    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;
    delete b ;
    delete d ;
}

/*-----------------------------------------------------------
-------------test5 OLD ------------
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 <-----------WHY
I am exam class sushma
I am exam class sushma <--------------WHY
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma

-------------test5 NOW ------------
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 cs32  97 <--- SOLVED
sushma  99 100 cs32  97 <--- SOLVED
I am exam class sushma
I am cs32exam class cs32 <--- SOLVED
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma
-----------------------------------------------------------*/
void test5() {
    cout << "-------------test5------------\n" ;
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;
```

```
    exam* b = &bi ;
    cout << *b << endl ;
    exam* d = &di ; /* Dynamic binding */
    cout << *d << endl ;
    d->print(cout) ; cout << endl ;
    b->who_am_i() ;
    d->who_am_i() ;
    d->exam::who_am_i() ;
}

/*-------------------------------------------------------
-------------test6 OLD ------------
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 <-----------WHY
I am exam class sushma
I am exam class sushma <--------------WHY
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma

-------------test6 NOW ------------
exam constructor sushma
exam constructor sushma
cs32 exam constructor cs32
sushma  52 80
sushma  99 100 cs32  97 <--- SOLVED
sushma  99 100 cs32  97 <--- SOLVED
I am exam class sushma
I am cs32exam class cs32 <--- SOLVED
I am exam class sushma
cs32exam destructor cs32
exam destructor sushma
exam destructor sushma
-------------------------------------------------------*/
void test6() {
    cout << "-------------test6------------\n" ;
    exam bi("sushma",52,80) ;
    cs32exam di("sushma","cs32",99,100,97) ;
    exam& b = bi ;
    cout << b << endl ;
    exam& d = di ; /* Dynamic binding */
    cout << d << endl ;
    d.print(cout) ; cout << endl ;
    b.who_am_i() ;
    d.who_am_i() ;
    d.exam::who_am_i() ;
}

/*-------------------------------------------------------
exam constructor sushma
sushma  52 80
```


VAMSI KRISHNA VAMURTHY

```
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100 cs32  97
sushma  99 100 cs32  97
I am cs32exam class cs32 <-- Dynamic binding
I am cs32exam class cs32 <-- Dynamic binding
I am exam class sushma <-- Dynamic binding
-----------------------------------------------------------*/
void test7() {
   exam *b = new exam("sushma",52,80) ;
   cout << *b << endl ;
   exam *d = new cs32exam("sushma","cs32",99,100,97) ;
   cout << *d << endl ;
   d->print(cout) ; cout << endl ;
   d->pass_by_reference(d) ;
   d->pass_by_value(d) ;
   b->pass_by_value(b) ;
}

/*-------------------------------------------------------
main
-----------------------------------------------------------*/
int main() {
   test4() ;
   test5() ;
   test6() ;
   test7() ;
   return 0 ;
}

```

VASUDEVAMURTHY

8.16 Abstract class and concept of pure virtual function

Pure Virtual function and abstract class

```
class exam {
    virtual void compute_grade() const = 0 ;
}
class cs32exam : public exam {
    virtual void compute_grade() const {
        cout << "final grade is : " << midterm() + final() + _project << endl ;
    }
}

void test_pure_virtual_function() {
    //exam b = new exam("sushma") ; //'exam' : cannot instantiate abstract class
    cs32exam* c = new cs32exam("jag","cs320",99,100,97) ;
    cout << *c << endl ;
    c->print(cout) ;cout << endl ;
    exam *d = new cs32exam("sushma","cs32",99,100,97) ;
    cout << *d << endl ;
    d->print(cout) ;cout << endl ;
    d->who_am_i() ;
    d->compute_grade() ;
    delete c ;
    delete d ;
}
```

```
exam constructor jag
cs32 exam constructor cs320
jag  99 100 cs320  97
jag  99 100 cs320  97
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100
sushma  99 100 cs32  97
I am cs32exam class cs32
final grade is : 296
cs32exam destructor cs320
exam destructor jag
cs32exam destructor cs32
exam destructor sushma
```

Figure 8.26: Abstract class and pure virtual function

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: purevirtual.cpp
----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
class exam
-------------------------------------------------------*/
class exam {
public:
   exam(const string& s, int m = 0, int f = 0):_sname(s),_midterm(m),_final(f) {
      cout << "exam constructor " << s << endl ;
   }
   virtual ~exam() { cout << "exam destructor " << _sname << endl ; }
   exam(const exam& e):_sname(e._sname), _midterm(e._midterm),_final(e._final) {
      cout << "exam copy constructor " << e._sname << endl ;
   }
   exam& operator=(const exam& e) {
      cout << "exam = operator " << e._sname << endl;
      if (&e != this) {
         _sname = e._sname ;
         _midterm = e._midterm ;
         _final = e._final ;
      }
      return *this ;
   }
   friend ostream& operator<<(ostream& o, const exam& s) {
      o << s._sname << "  " << s._midterm << " " << s._final << " " ;
      return o ;
   }
   virtual ostream& print(ostream& o) const {
      o << *this ;
      return o ;
   }
   virtual void who_am_i() const {
      cout << "I am exam class " << _sname << endl ;
   }
   virtual void compute_grade() const = 0 ; //PURE VIRTIAL FUNCTION - ABSTRACT CLASS
   int midterm() const {return _midterm ; }
   int final() const {return _final ;}

private:
   string _sname ;
   int _midterm ;
   int _final ;
};

/*-------------------------------------------------------
cs32 exam
-------------------------------------------------------*/
```


VASUDEVAMURTHY

```
class cs32exam : public exam {
public:
   cs32exam(const string& s,const string& e, int m = 0, int f = 0,int p = 0) :
     exam(s,m,f)/*Note this call*/,_examname(e),_project(p) {
        cout << "cs32 exam constructor " << e << endl ;
     }
     ~cs32exam() { cout << "cs32exam destructor " << _examname << endl ; }
     cs32exam(const cs32exam& e):exam(e)/* NOTE */,_examname(e._examname),_project(e._project)  {
        cout << "cs32exam copy constructor " << e._examname << endl ;
     }
     cs32exam& operator=(const cs32exam& e) {
        if (&e != this) {
           cout << "cs32exam = operator " << e._examname << endl ;
           exam::operator=(e) ; //Note this call.  base of e is copied base of this e
           _examname = e._examname ;
           _project = e._project;
        }
        return *this ;
     }
     friend ostream& operator<<(ostream& o, const cs32exam& s) {
        const exam* e = &s ;
        o << *e ;
        o << s._examname<< "  " << s._project<< " ";
        return o ;
     }
     virtual ostream& print(ostream& o) const {
        o << *this ;
        return o ;
     }
     void who_am_i() const {
        cout << "I am cs32exam class " << _examname << endl ;
     }
     virtual void compute_grade() const {
        cout << "final grade is : " << midterm() + final() + _project << endl ;
     }
private:
   string _examname ;
   int _project ;
} ;

/*-------------------------------------------------------
exam constructor jag
cs32 exam constructor cs320
jag  99 100 cs320  97
jag  99 100 cs320  97
exam constructor sushma
cs32 exam constructor cs32
sushma  99 100
sushma  99 100 cs32  97
I am cs32exam class cs32
final grade is : 296
cs32exam destructor cs320
exam destructor jag
cs32exam destructor cs32
exam destructor sushma
```


VASUDEVAMURTHY

```
111 ----------------------------------------------------------*/
112 void test_pure_virtual_function() {
113     //exam b = new exam("sushma") ; //'exam' : cannot instantiate abstract class
114     cs32exam* c = new cs32exam("jag","cs320",99,100,97) ;
115     cout << *c << endl ;
116     c->print(cout) ;cout << endl ;
117     exam *d = new cs32exam("sushma","cs32",99,100,97) ;
118     cout << *d << endl ;
119     d->print(cout) ;cout << endl ;
120     d->who_am_i() ;
121     d->compute_grade() ;
122     delete c ;
123     delete d ;
124 }
125
126 /*----------------------------------------------------------
127 main
128 ----------------------------------------------------------*/
129 int main() {
130     test_pure_virtual_function() ;
131     return 0 ;
132 }
133
```

VASUDEVAMURTHY

8.17 Casting from derived to base and base to derived

Figure 8.27: Casting from derived to base

VASUDEVAMURTHY

Casting base class to derived class

base <= derived

When base is put to derived, what happens to extra space?

```
cs32exam derived = base ;
'initializing' : cannot convert from 'exam' to 'cs32exam
```

```
cs32exam(const exam& e) :exam(e),_examname("hole"),_project(-1) {
    cout << "cs32exam conversion constructor " << "hole" << endl ;
}
```

1. This constructor must be written in derived class
2. base class DOES NOT know about derived class
3. This constructor MUST TAKE base class as a parameter
4. It should take care of HOLES

```
void test_base_2_derived() {
    exam base("sushma",99,100) ;
    base.print(cout) ;cout << endl ;
    cs32exam derived = base ;
    derived.print(cout) ;cout << endl ;
}
```

```
exam constructor sushma
sushma  99 100
exam copy constructor sushma
cs32exam conversion constructor hole
sushma  99 100 hole  -1
cs32exam destructor hole
exam destructor sushma
exam destructor sushma
```

```
cs32exam(const exam& e) :exam(e),_examname("hole"),_project(-1) {

}
```

Must take base class Note the call: exam(e) HOLES fixed

Figure 8.28: Casting from base to derived

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: casting.cpp
----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
class exam
-------------------------------------------------------*/
class exam {
public:
   exam(const string& s, int m = 0, int f = 0):_sname(s),_midterm(m),_final(f) {
      cout << "exam constructor " << s << endl ;
   }
   virtual ~exam() { cout << "exam destructor " << _sname << endl ; }
   exam(const exam& e):_sname(e._sname), _midterm(e._midterm),_final(e._final) {
      cout << "exam copy constructor " << e._sname << endl ;
   }
   exam& operator=(const exam& e) {
      cout << "exam = operator " << e._sname << endl;
      if (&e != this) {
         _sname = e._sname ;
         _midterm = e._midterm ;
         _final = e._final ;
      }
      return *this ;
   }
   friend ostream& operator<<(ostream& o, const exam& s) {
      o << s._sname << "  " << s._midterm << " " << s._final << " " ;
      return o ;
   }
   virtual ostream& print(ostream& o) const {
      o << *this ;
      return o ;
   }

private:
   string _sname ;
   int _midterm ;
   int _final ;
};

/*-------------------------------------------------------
cs32 exam
-------------------------------------------------------*/
class cs32exam : public exam {
public:
   cs32exam(const string& s,const string& e, int m = 0, int f = 0,int p = 0) :
     exam(s,m,f)/*Note this call*/,_examname(e),_project(p) {
        cout << "cs32 exam constructor " << e << endl ;
     }
```


VASUDEVAMURTHY

```
    ~cs32exam() { cout << "cs32exam destructor " << _examname << endl ; }
    cs32exam(const cs32exam& e):exam(e)/* NOTE */,_examname(e._examname),_project(e._project)  {
       cout << "cs32exam copy constructor " << e._examname << endl ;
    }
    cs32exam& operator=(const cs32exam& e) {
       if (&e != this) {
          cout << "cs32exam = operator " << e._examname << endl ;
          exam::operator=(e) ; //Note this call.  base of e is copied base of this e
          _examname = e._examname ;
          _project = e._project;
       }
       return *this ;
    }
    friend ostream& operator<<(ostream& o, const cs32exam& s) {
       const exam* e = &s ;
       o << *e ;
       o << s._examname<< "  " << s._project<< " ";
       return o ;
    }
    virtual ostream& print(ostream& o) const {
       o << *this ;
       return o ;
    }
    //Constructor that takes a base class and generate derived class
    cs32exam(const exam& e) :exam(e),_examname("hole"),_project(-1) {
       cout << "cs32exam conversion constructor " << "hole" << endl ;
    }

private:
   string _examname ;
   int _project ;
} ;

/*-------------------------------------------------------
derived is >= base
So derived can always put to base
derived class portion is thrown away keeping only base portion

exam constructor sushma
cs32 exam constructor cs320
sushma  99 100 cs320  97
exam copy constructor sushma
sushma  99 100
exam destructor sushma
cs32exam destructor cs320
exam destructor sushma
-------------------------------------------------------*/
void test_derived_2_base() {
   cs32exam derived("sushma","cs320",99,100,97) ;
   derived.print(cout) ;cout << endl ;
   exam base = derived ;
   base.print(cout) ;cout << endl ;
}

/*-------------------------------------------------------
```

```
base <= derived
So when base is put to derived, what happens to the extra space? What data it will have?
 'initializing' : cannot convert from 'exam' to 'cs32exam
So you need to have a converter which takes base and convert to derived

exam constructor sushma
sushma  99 100
exam copy constructor sushma
cs32exam conversion constructor hole
sushma  99 100 hole  -1
cs32exam destructor hole
exam destructor sushma
exam destructor sushma
-----------------------------------------------------------*/
void test_base_2_derived() {
    exam base("sushma",99,100) ;
    base.print(cout) ;cout << endl ;
    cs32exam derived = base ;
    derived.print(cout) ;cout << endl ;
}

/*-----------------------------------------------------------
main
-----------------------------------------------------------*/
int main() {
    test_derived_2_base() ;
    test_base_2_derived();
    return 0 ;
}

```

VASUDEVAMURTHY

8.18 Implementing polymorphism without using virtual

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
Filename: implement_virtual.cpp
-----------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

#ifdef WIN32
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#endif

/*-------------------------------------------------------
Polymorphism using Virtual
-------------------------------------------------------*/
class animal {
public:
  animal() {cout << "animal constructor\n";}
  void where_am_i() { cout << "I am In Africa\n" ; } //NON VIRTUAL
  virtual void who_am_i() { cout << "I am an animal\n" ; } //VIRTUAL
  virtual ~animal() {cout << "animal destructor\n";} //VIRTUAL.
};


class lion:public animal {
public:
  lion() {cout << "lion constructor\n";}
  virtual void who_am_i() { cout << "I am a lion\n" ; }
  ~lion() {cout << "lion destructor\n";}
};


/*-------------------------------------------------------

-------------------------------------------------------*/
void test_virtual_polymorphism() {
  animal* a[2] ;
  a[0]  = new animal;
  a[1] = new lion ;
  for (int i = 0; i < 2; i++) {
    a[i]->where_am_i() ;
    a[i]->who_am_i() ;
  }
  for (int i = 0; i < 2; i++) {
    delete a[i] ;
  }
}

/*-------------------------------------------------------
Polymorphism WITHOUT using Virtual
-------------------------------------------------------*/
class ANIMAL;
typedef void(ANIMAL::*ANIMAL_PTR)(void) ;

class LION;
typedef void(LION::*LION_PTR)(void) ;


class ANIMAL {
public:
  static ANIMAL_PTR animal_vtable ;
  //Made static to show that vtable is ONLY one for each CLASS
  //Even though this class is instantiated million times.
  ANIMAL_PTR* vptr ; //Note global. Can access from main for easy coding
```


VASUDEVAMURTHY

```

  ANIMAL() {cout << "ANIMAL constructor\n";vptr = &animal_vtable; }
  void where_am_i() { cout << "I am In Africa\n" ; }
  void who_am_i() { cout << "I am an ANIMAL\n" ; }
  ~ANIMAL() {cout << "ANIMAL destructor\n";}
};

class LION:public ANIMAL {
public:
  static LION_PTR lion_vtable ;
  //Made static to show that vtable is ONLY one for each CLASS
  //Even though this class is instantiated million times.
  LION() {cout << "LION constructor\n"; vptr = (ANIMAL_PTR*) &lion_vtable;}
  void who_am_i() { cout << "I am a LION\n" ; }
  ~LION() {cout << "LION destructor\n";}
};

/*-------------------------------------------------------
Initalize all static now
PURPOSEFULLY ~distructor is NOT made virtual
-------------------------------------------------------*/
ANIMAL_PTR ANIMAL::animal_vtable = {&ANIMAL::who_am_i};
LION_PTR LION::lion_vtable = {&LION::who_am_i};

void test_non_virtual_polymorphism() {
  ANIMAL* a[2] ;
  a[0] = new ANIMAL ;
  a[1] = new LION ;
  for (int i = 0; i < 2; i++) {
    a[i]->where_am_i() ; //Static binding
    ANIMAL_PTR ptr = *(a[i]->vptr) ;
    ((a[i])->*(ptr)) () ; //Dynamic binding
  }
  for (int i = 0; i < 2; i++) {
    delete a[i] ;
  }
}

/*-------------------------------------------------------
main

animal constructor
animal constructor
lion constructor
I am In Africa
I am an animal
I am In Africa
I am a lion
animal destructor
lion destructor
animal destructor
===================
ANIMAL constructor
ANIMAL constructor
LION constructor
I am In Africa
I am an ANIMAL
I am In Africa
I am a LION
ANIMAL destructor
<<-------------------- BUG HERE ON PURPOSE
ANIMAL destructor


-------------------------------------------------------*/
void main() {
```


VASUDEVAMURTHY

```
  test_virtual_polymorphism() ;
  test_non_virtual_polymorphism();
}


//EOF

```

VASUDEVAMURTHY

8.19 Problem set

Problem 8.19.1. Implement *employee* class as explained in figure 8.29. The driver program *test.cpp* is given. All the tests must pass. The program must use a static data member called *show* in the *employee* class. When this variable is *true*, the program must print the call of constructors, destructors, copy constructors and equal operators.

email the following files, along with copy of the output of the program.

```
employee.h
employee.cpp
salariedemployee.h
salariedemployee.cpp
commissionemployee.h
commissionemployee.cpp
baseplusscommissionemployee.h
baseplusscommissionemployee.cpp
output of the program as a doc file
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
Filename: test.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
            NOTHING CAN BE CHANGED BELOW
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "employee.h"
#include "salariedemployee.h"
#include "commissionemployee.h"
#include "basepluscommissionemployee.h"

/*-----------------------------------------------------------------
testEmployee
-----------------------------------------------------------------*/
void testEmployee() {
  employee e1("jag", "vasudevamurthy", "678-90");
  cout << e1 << endl;
  employee e2("bob", "ericission", "178-90");
  cout << e2 << endl;
  employee e3(e1);
  cout << e3 << endl;
  e2 = e3;
  cout << e3 << endl;
  cout << e2 << endl;
}

/*-----------------------------------------------------------------
testSalariedemployee
-----------------------------------------------------------------*/
void testSalariedemployee() {
  salariedemployee e1("jag", "vasudevamurthy", "678-90", "Xilinx", 56000);
  cout << e1 << endl;
  salariedemployee e2("bob", "ericission", "178-90", "ATT", 3456000);
  cout << e2 << endl;
  salariedemployee e3(e1);
  cout << e3 << endl;
  e2 = e3;
  cout << e3 << endl;
  cout << e2 << endl;
}

/*-----------------------------------------------------------------
test Commission employee
-----------------------------------------------------------------*/
```

```
53   void testCommissionemployee() {
54     commissionemployee e1("jag", "vasudevamurthy", "678-90", "BELL", 560);
55     cout << e1 << endl;
56     commissionemployee e2("bob", "ericission", "178-90", "UCSC", 345);
57     cout << e2 << endl;
58     commissionemployee e3(e1);
59     cout << e3 << endl;
60     e2 = e3;
61     cout << e3 << endl;
62     cout << e2 << endl;
63   }
64
65   /*-------------------------------------------------------------
66   test Basepluscommissionemployee
67   ----------------------------------------------------------------*/
68   void testBasepluscommissionemployee() {
69     basepluscommissionemployee e1("jag", "vasudevamurthy", "678-90", "BELL",
     560, "abc", 89);
70     cout << e1 << endl;
71     basepluscommissionemployee e2("bob", "ericission", "178-90", "UCSC", 345,
      "xyz", 90);
72     cout << e2 << endl;
73     basepluscommissionemployee e3(e1);
74     cout << e3 << endl;
75     e2 = e3;
76     cout << e3 << endl;
77     cout << e2 << endl;
78   }
79
80   /*-------------------------------------------------------------
81   If this fails means you don't understand C++
82   You wasted 1000 dollars
83   ----------------------------------------------------------------*/
84   void testBasepluscommissionemployeeYouAreBusted() {
85     const int N = 5 ;
86     employee* e[N] ;
87     for (int i = 0; i < 5; ++i) {
88       char fname[10] ;
89       char lname[10] ;
90       char ssn[10] ;
91       char a[2] ;
92       a[0] = i + '0' ;
93       a[1] = '\0' ;
94       strcpy(fname,"jag") ;
95       strcat(fname,a) ;
96       strcpy(lname,"Vas") ;
97       strcat(lname,a) ;
98       strcpy(ssn,"46-78-") ;
99       strcat(ssn,a) ;
100      char w1[10] ;
101      strcpy(w1,"BELL") ;
102      strcat(w1,a) ;
```

```
      char w2[10] ;
      strcpy(w2,"xyz") ;
      strcat(w2,a) ;
      e[i] = new baseplusscommissionemployee(fname,lname,ssn,w1,100+i,w2,878+i);
      const employee& ei = *(e[i]);
      cout << ei << endl;
    }
    for (int i = 0; i < N; ++i) {
      cout << *(e[i]) << endl ;
    }
    *(e[0]) = *(e[1]) = *(e[2]) ;
    for (int i = 0; i < N; ++i) {
      cout << *(e[i]) << endl ;
    }
    for (int i = 0; i < N; ++i) {
      delete e[i] ;
    }
  }

  /*-----------------------------------------------------------------
  test
  -----------------------------------------------------------------*/
  void polymorphism() {
    vector<employee *> v;
    {
      employee* e1 = new employee("jag", "vasudevamurthy", "678-90");
      v.push_back(e1);
      employee* e2 = new employee("bob", "ericission", "178-90");
      v.push_back(e2);
    }
    {
      salariedemployee*  e1 = new salariedemployee("alex", "wong", "678-89", "Men", 24);
      v.push_back(e1);
      salariedemployee* e2 = new salariedemployee("bob", "john", "478-90", "XYX", 345);
      v.push_back(e2);
    }

    {
      commissionemployee* e1 = new commissionemployee("hale", "tom", "589-78", "xil", 560000);
      v.push_back(e1);
      commissionemployee* e2 = new commissionemployee("jon", "snith", "478-90", "UCB", 9);
      v.push_back(e2);
    }

    {
      baseplusscommissionemployee* e1 = new baseplusscommissionemployee("amy", "vmay", "878-90", "synp", 160, "nvida", 809);
```

```
void polymorphism() {
  vector<employee *> v;
  {
    employee* e1 = new employee("jag", "vasudevamurthy", "678-90");
    v.push_back(e1);
    employee* e2 = new employee("bob", "ericission", "178-90");
    v.push_back(e2);
  }
  {
    salariedemployee*  e1 = new salariedemployee("alex", "wong", "678-89", "Men", 24);
    v.push_back(e1);
    salariedemployee* e2 = new salariedemployee("bob", "john", "478-90", "XYX", 345);
    v.push_back(e2);
  }

  {
    commissionemployee* e1 = new commissionemployee("hale", "tom", "589-78", "xil", 560000);
    v.push_back(e1);
    commissionemployee* e2 = new commissionemployee("jon", "snith", "478-90", "UCB", 9);
    v.push_back(e2);
  }

  {
    basepluscommissionemployee* e1 = new basepluscommissionemployee("amy", "vmay", "878-90", "synp", 160, "nvida", 80
    v.push_back(e1);
    basepluscommissionemployee* e2 = new basepluscommissionemployee("fake", "guy", "878-60", "mu", 345, "spy", 190);
    v.push_back(e2);
  }

  {
    for (auto i = 0; i < v.size(); ++i) {
      cout << i << ": " << *(v[i]) << endl;
    }
  }
  {
    for (auto i : v) {
      delete i;
    }
  }
}
```

```
0: jag  vasudevamurthy 678-90
1: bob  ericission 178-90
2: alex  wong 678-89 Men  24
3: bob  john 478-90 XYX  345
4: hale  tom 589-78 xil  560000
5: jon  snith 478-90 UCB  9
6: amy  vmay 878-90 synp  160 nvida  809
7: fake  guy 878-60 mu  345 spy  190
```

Figure 8.29: Implementing inheritence and polymorphism

VASUDEVAMURTHY

```
In employee::employee constructor jag  vasudevamurthy 678-90
jag  vasudevamurthy 678-90
In employee::employee constructor bob  ericission 178-90
bob  ericission 178-90
In employee copy constructor jag  vasudevamurthy 678-90
jag  vasudevamurthy 678-90
In employee equal constructor jag  vasudevamurthy 678-90
jag  vasudevamurthy 678-90
jag  vasudevamurthy 678-90
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::employee constructor jag  vasudevamurthy 678-90
In salariedemployee::salariedemployee constructor jag  vasudevamurthy 678-90 Xilinx  56000
jag  vasudevamurthy 678-90 Xilinx  56000
In employee::employee constructor bob  ericission 178-90
In salariedemployee::salariedemployee constructor bob  ericission 178-90 ATT  3.456e+06
bob  ericission 178-90 ATT  3.456e+06
In employee copy constructor jag  vasudevamurthy 678-90
In salariedemployee copy constructor jag  vasudevamurthy 678-90 Xilinx  56000
jag  vasudevamurthy 678-90 Xilinx  56000
In employee equal constructor jag  vasudevamurthy 678-90 ATT  3.456e+06
In salariedemployee equal constructor jag  vasudevamurthy 678-90 Xilinx  56000
jag  vasudevamurthy 678-90 Xilinx  56000
jag  vasudevamurthy 678-90 Xilinx  56000
In salariedemployee::~salariedemployee distructor jag  vasudevamurthy 678-90 Xilinx  56000
In employee::~employee distructor jag  vasudevamurthy 678-90
In salariedemployee::~salariedemployee distructor jag  vasudevamurthy 678-90 Xilinx  56000
In employee::~employee distructor jag  vasudevamurthy 678-90
In salariedemployee::~salariedemployee distructor jag  vasudevamurthy 678-90 Xilinx  56000
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::employee constructor jag  vasudevamurthy 678-90
In commissionemployee::commissionemployee constructor jag  vasudevamurthy 678-90 BELL  560
jag  vasudevamurthy 678-90 BELL  560
In employee::employee constructor bob  ericission 178-90
In commissionemployee::commissionemployee constructor bob  ericission 178-90 UCSC  345
bob  ericission 178-90 UCSC  345
In employee copy constructor jag  vasudevamurthy 678-90
In commissionemployee copy constructor jag  vasudevamurthy 678-90 BELL  560
jag  vasudevamurthy 678-90 BELL  560
In employee equal constructor jag  vasudevamurthy 678-90 UCSC  345
In commissionemployee equal constructor jag  vasudevamurthy 678-90 BELL  560
jag  vasudevamurthy 678-90 BELL  560
jag  vasudevamurthy 678-90 BELL  560
In commissionemployee::~commissionemployee distructor jag  vasudevamurthy 678-90 BELL  560
In employee::~employee distructor jag  vasudevamurthy 678-90
In commissionemployee::~commissionemployee distructor jag  vasudevamurthy 678-90 BELL  560
In employee::~employee distructor jag  vasudevamurthy 678-90
```


VASUDEVAMURTHY

```
In commissionemployee::~commissionemployee distructor jag  vasudevamurthy 678-90 BELL  560
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::employee constructor jag  vasudevamurthy 678-90
In commissionemployee::commissionemployee constructor jag  vasudevamurthy 678-90 BELL  560
In basepluscommissionemployee::basepluscommissionemployee constructor jag  vasudevamurthy 678-
90 BELL  560 abc  89
jag  vasudevamurthy 678-90 BELL  560 abc  89
In employee::employee constructor bob  ericission 178-90
In commissionemployee::commissionemployee constructor bob  ericission 178-90 UCSC  345
In basepluscommissionemployee::basepluscommissionemployee constructor bob  ericission 178-90
UCSC  345 xyz  90
bob  ericission 178-90 UCSC  345 xyz  90
In employee copy constructor jag  vasudevamurthy 678-90
In commissionemployee copy constructor jag  vasudevamurthy 678-90 BELL  560
In basepluscommissionemployee copy constructor jag  vasudevamurthy 678-90 BELL  560 abc  89
jag  vasudevamurthy 678-90 BELL  560 abc  89
In employee equal constructor jag  vasudevamurthy 678-90 UCSC  345 xyz  90
In commissionemployee equal constructor jag  vasudevamurthy 678-90 BELL  560 xyz  90
In basepluscommissionemployee equal constructor jag  vasudevamurthy 678-90 BELL  560 abc  89
jag  vasudevamurthy 678-90 BELL  560 abc  89
jag  vasudevamurthy 678-90 BELL  560 abc  89
In basepluscommissionemployee::~basepluscommissionemployee distructor jag  vasudevamurthy 678-
90 BELL  560 abc  89
In commissionemployee::~commissionemployee distructor jag  vasudevamurthy 678-90 BELL  560
In employee::~employee distructor jag  vasudevamurthy 678-90
In basepluscommissionemployee::~basepluscommissionemployee distructor jag  vasudevamurthy 678-
90 BELL  560 abc  89
In commissionemployee::~commissionemployee distructor jag  vasudevamurthy 678-90 BELL  560
In employee::~employee distructor jag  vasudevamurthy 678-90
In basepluscommissionemployee::~basepluscommissionemployee distructor jag  vasudevamurthy 678-
90 BELL  560 abc  89
In commissionemployee::~commissionemployee distructor jag  vasudevamurthy 678-90 BELL  560
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::employee constructor jag0  Vas0 46-78-0
In commissionemployee::commissionemployee constructor jag0  Vas0 46-78-0 BELL0  100
In basepluscommissionemployee::basepluscommissionemployee constructor jag0  Vas0 46-78-0 BELL0
100 xyz0  878
jag0  Vas0 46-78-0 BELL0  100 xyz0  878
In employee::employee constructor jag1  Vas1 46-78-1
In commissionemployee::commissionemployee constructor jag1  Vas1 46-78-1 BELL1  101
In basepluscommissionemployee::basepluscommissionemployee constructor jag1  Vas1 46-78-1 BELL1
101 xyz1  879
jag1  Vas1 46-78-1 BELL1  101 xyz1  879
In employee::employee constructor jag2  Vas2 46-78-2
In commissionemployee::commissionemployee constructor jag2  Vas2 46-78-2 BELL2  102
In basepluscommissionemployee::basepluscommissionemployee constructor jag2  Vas2 46-78-2 BELL2
102 xyz2  880
jag2  Vas2 46-78-2 BELL2  102 xyz2  880
```


VASUDEVAMURTHY

```
In employee::employee constructor jag3  Vas3 46-78-3
In commissionemployee::commissionemployee constructor jag3  Vas3 46-78-3 BELL3  103
In basepluscommissionemployee::basepluscommissionemployee constructor jag3  Vas3 46-78-3 BELL3
103 xyz3  881
jag3  Vas3 46-78-3 BELL3  103 xyz3  881
In employee::employee constructor jag4  Vas4 46-78-4
In commissionemployee::commissionemployee constructor jag4  Vas4 46-78-4 BELL4  104
In basepluscommissionemployee::basepluscommissionemployee constructor jag4  Vas4 46-78-4 BELL4
104 xyz4  882
jag4  Vas4 46-78-4 BELL4  104 xyz4  882
jag0  Vas0 46-78-0 BELL0  100 xyz0  878
jag1  Vas1 46-78-1 BELL1  101 xyz1  879
jag2  Vas2 46-78-2 BELL2  102 xyz2  880
jag3  Vas3 46-78-3 BELL3  103 xyz3  881
jag4  Vas4 46-78-4 BELL4  104 xyz4  882
In employee equal constructor jag2  Vas2 46-78-2 BELL1  101 xyz1  879
In employee equal constructor jag2  Vas2 46-78-2 BELL0  100 xyz0  878
jag2  Vas2 46-78-2 BELL0  100 xyz0  878
jag2  Vas2 46-78-2 BELL1  101 xyz1  879
jag2  Vas2 46-78-2 BELL2  102 xyz2  880
jag3  Vas3 46-78-3 BELL3  103 xyz3  881
jag4  Vas4 46-78-4 BELL4  104 xyz4  882
In basepluscommissionemployee::~basepluscommissionemployee distructor jag2  Vas2 46-78-2 BELL0
100 xyz0  878
In commissionemployee::~commissionemployee distructor jag2  Vas2 46-78-2 BELL0  100
In employee::~employee distructor jag2  Vas2 46-78-2
In basepluscommissionemployee::~basepluscommissionemployee distructor jag2  Vas2 46-78-2 BELL1
101 xyz1  879
In commissionemployee::~commissionemployee distructor jag2  Vas2 46-78-2 BELL1  101
In employee::~employee distructor jag2  Vas2 46-78-2
In basepluscommissionemployee::~basepluscommissionemployee distructor jag2  Vas2 46-78-2 BELL2
102 xyz2  880
In commissionemployee::~commissionemployee distructor jag2  Vas2 46-78-2 BELL2  102
In employee::~employee distructor jag2  Vas2 46-78-2
In basepluscommissionemployee::~basepluscommissionemployee distructor jag3  Vas3 46-78-3 BELL3
103 xyz3  881
In commissionemployee::~commissionemployee distructor jag3  Vas3 46-78-3 BELL3  103
In employee::~employee distructor jag3  Vas3 46-78-3
In basepluscommissionemployee::~basepluscommissionemployee distructor jag4  Vas4 46-78-4 BELL4
104 xyz4  882
In commissionemployee::~commissionemployee distructor jag4  Vas4 46-78-4 BELL4  104
In employee::~employee distructor jag4  Vas4 46-78-4
In employee::employee constructor jag  vasudevamurthy 678-90
In employee::employee constructor bob  ericission 178-90
In employee::employee constructor alex  wong 678-89
In salariedemployee::salariedemployee constructor alex  wong 678-89 Men  24
In employee::employee constructor bob  john 478-90
In salariedemployee::salariedemployee constructor bob  john 478-90 XYX  345
```


VASUDEVAMURTHY

```
In employee::employee constructor hale  tom 589-78
In commissionemployee::commissionemployee constructor hale  tom 589-78 xil  560000
In employee::employee constructor jon  snith 478-90
In commissionemployee::commissionemployee constructor jon  snith 478-90 UCB  9
In employee::employee constructor amy  vmay 878-90
In commissionemployee::commissionemployee constructor amy  vmay 878-90 synp  160
In basepluscommissionemployee::basepluscommissionemployee constructor amy  vmay 878-90 synp
160 nvida  809
In employee::employee constructor fake  guy 878-60
In commissionemployee::commissionemployee constructor fake  guy 878-60 mu  345
In basepluscommissionemployee::basepluscommissionemployee constructor fake  guy 878-60 mu  345
spy  190
0: jag  vasudevamurthy 678-90
1: bob  ericission 178-90
2: alex  wong 678-89 Men  24
3: bob  john 478-90 XYX  345
4: hale  tom 589-78 xil  560000
5: jon  snith 478-90 UCB  9
6: amy  vmay 878-90 synp  160 nvida  809
7: fake  guy 878-60 mu  345 spy  190
In employee::~employee distructor jag  vasudevamurthy 678-90
In employee::~employee distructor bob  ericission 178-90
In salariedemployee::~salariedemployee distructor alex  wong 678-89 Men  24
In employee::~employee distructor alex  wong 678-89
In salariedemployee::~salariedemployee distructor bob  john 478-90 XYX  345
In employee::~employee distructor bob  john 478-90
In commissionemployee::~commissionemployee distructor hale  tom 589-78 xil  560000
In employee::~employee distructor hale  tom 589-78
In commissionemployee::~commissionemployee distructor jon  snith 478-90 UCB  9
In employee::~employee distructor jon  snith 478-90
In basepluscommissionemployee::~basepluscommissionemployee distructor amy  vmay 878-90 synp
160 nvida  809
In commissionemployee::~commissionemployee distructor amy  vmay 878-90 synp  160
In employee::~employee distructor amy  vmay 878-90
In basepluscommissionemployee::~basepluscommissionemployee distructor fake  guy 878-60 mu  345
spy  190
In commissionemployee::~commissionemployee distructor fake  guy 878-60 mu  345
In employee::~employee distructor fake  guy 878-60
```

VASUDEVAMURTHY

VASUDEVAMURTHY

Chapter 9

Namespaces

9.1 Introduction

9.2 What is the problem?

```
/*-------------------
n1.h
--------------------*/
#ifndef N1_H
#define N1_H

/*-------------------------
Declaration of sqrt class
---------------------------*/
class sqrt{
public:
  sqrt() {cout << "I am in sqrt of file n1.h" << endl;}

private:

};

#endif
```

```
/*-------------------
n2.h
--------------------*/
#ifndef N2_H
#define N2_H

/*-------------------------
Declaration of sqrt class
---------------------------*/
class sqrt{
public:
  sqrt() {cout << "I am in sqrt of file n2.h" << endl;}

private:

};

#endif
```

```
main.cpp

 #include "n1.h"
 #include "n2.h"
 main() {

 }
```

Figure 9.1: Problem

VASUDEVAMURTHY

9.3 Need for namespace

```
/*-------------------
n1.h
--------------------*/
#ifndef N1_H
#define N1_H

/*-------------------------
Declaration of sqrt class
----------------------------*/
namespace N1 {
 class sqrt{
 public:
  sqrt() {cout << "I am in sqrt of file n1.h" << endl;}

 private:
 };
}
#endif
```

```
/*-------------------
n2.h
--------------------*/
#ifndef N2_H
#define N2_H

/*-------------------------
Declaration of sqrt class
----------------------------*/
namespace N2 {
class sqrt{
 public:
  sqrt() {cout << "I am in sqrt of file n2.h" << endl;}

 private:
 };
}
#endif
```

```
main.cpp

 #include "n1.h"
 #include "n2.h"
 main() {
    N1::sqrt a ;
 }  N2::sqrt b ;
```

Namespace provide a solution to the problem of duplicate names in the global space

Figure 9.2: Need for namespace

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: n1.h
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#ifndef N1_H
#define N1_H

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

#ifdef _WIN32
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#endif

namespace N1 {
  //A variable
  int x ;
  //A function
  void hello() {cout << "I am hello of namespace N1\n" ;}
  void f1() ;
  class arith{
  public:
    arith() {cout << "I am in class arith of file n1.h" << endl ;}
    void f2() ;
  private:

  };
}

/*----------------------------------------------------------------
To show namespace functions can be written outside
----------------------------------------------------------------*/
void N1::f1() {
  cout << "I am in N1::f1()\n" ;
}
void N1::arith::f2() {
  cout << "I am in N1::arith::f2()\n" ;
}

/*----------------------------------------------------------------
To show namespace can be extended in same file or in different file
----------------------------------------------------------------*/
namespace N1 {
  //A variable
  int x2 ;
}

#endif
//EOF
```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
file: n2.h
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#ifndef N2_H
#define N2_H

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

#ifdef _WIN32
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#endif

namespace N2 {
  //A variable
  int x ;
  //A function
  void hello() {cout << "I am hello of namespace N2\n"; }
  class arith{
  public:
    arith() {cout << "I am in class arith of file n2.h" << endl ;}
  private:

  };
}


#endif
//EOF

```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2013 Author: Jagadeesh Vasudevamurthy
Filename: n1.cpp
----------------------------------------------------------------*/
#include <iostream>
#include <cassert>
#include <string>
using namespace std;

#ifdef _WIN32
#include "vld.h"  //Comment this line, if you have NOT installed Visual leak detector
#endif

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "n1.h"
#include "n2.h"

/*-------------------------------------------------------
Gloal variables
-------------------------------------------------------*/
int y = 80 ;

/*-------------------------------------------------------
UNARY SCOPE RESOLUTION OPERATOR:
::x = 9089 ;
It tells the compiler to look RHS varaible(x) in GLOBAL SCOPE
BINARY SCOPE RESOLUTION OPERATOR
N1::x= 40 ;
Look for the RHS variable(x) in the NAMESPACE of the LHS(N1)
-------------------------------------------------------*/
void accessing_name_space_directly() {
  //accessing variable
  N1::x= 40 ; //BINARY SCOPE RESOLUTION OPERATOR
  N2::x= 40 ;
  //accessing function
  N1::hello() ;
  N2::hello() ;
  //accessing class
  N1::arith a;
  N2::arith b ;

  int y = 90 ;
  cout << "Local y = " << y << endl ;

  //Without :: you could not have accessed global y
  ::y = 9089 ; //UNARY SCOPE RESOLUTION OPERATOR
  cout << "global y = " << ::y << endl ;

  N1::f1() ;
  N1::x2 = 80 ;
  a.f2() ;
}

/*-------------------------------------------------------

-------------------------------------------------------*/
void accessing_name_space_by_using() {
  {
    using namespace N1;
    //accessing variable
    x= 40 ; //BINARY SCOPE RESOLUTION OPERATOR
    N2::x= 80 ;
    cout << "N1 x = " << x << endl ;
    cout << "N2 x = " << N2::x << endl ;
```


VASUDEVAMURTHY

```
    //accessing function
    hello() ;
    N2::hello() ;
    //accessing class
    arith a;
    N2::arith b ;
    //accessing global
    cout << "global y = " << y << endl ;
    cout << "global y = " << ::y << endl ;
  }
  {
    //Bring N2
    using namespace N2;
    //accessing variable
    x= 40 ; //BINARY SCOPE RESOLUTION OPERATOR
    N1::x= 80 ;
    cout << "N2 x = " << x << endl ;
    cout << "N1 x = " << N1::x << endl ;
    //accessing function
    hello() ;
    N1::hello() ;
    //accessing class
    arith a;
    N1::arith b ;
    //accessing global
    cout << "global y = " << y << endl ;
    cout << "global y = " << ::y << endl ;
  }
}

/*-------------------------------------------------------
Gloal variables again
-------------------------------------------------------*/
int x = 80 ;

/*-------------------------------------------------------
WILL NOT COMPILE

'x' : ambiguous symbol
could be 'n1.cpp(96) : int x'  (GLOBAL)
or         n1.h(23) :  int N1::x'
-------------------------------------------------------*/
#if 0
void ambigious_situation() {
  using namespace N1;
  x= 40 ;
  N2::x= 80 ;
}
#endif

void ambigious_situation_fixed() {
  using namespace N1;
  ::x= 40 ;
  N1::x = 9 ;
  N2::x= 80 ;
}

namespace UCSC {
  const char* name = "UCSC EXTENSION" ;
}
/*-------------------------------------------------------

-------------------------------------------------------*/
void you_are_using_already() {
  //If you comment using namespace std;
  std::cout << "Hello World \n" ;
```


VAMUDEVAMURTHY

```
  //If you uncomment using namespace std;
  cout << "Hello World \n" ;
  cout << UCSC::name << endl ;
  //cout << name ; //'name' : undeclared identifier
  using namespace UCSC;
  cout << name << endl ;
}


/*-------------------------------------------------------
main
-------------------------------------------------------*/
void main() {
  accessing_name_space_directly() ;
  accessing_name_space_by_using() ;
  ambigious_situation_fixed();
  you_are_using_already() ;
}


//EOF
```

VASUDEVAMURTHY

VASUDEVAMURTHY

Chapter 10

Template

10.1 Introduction

10.2 Macros

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: macro.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
int minimum
----------------------------------------------------------------*/
static void minimum(int i, int j) {
  cout << " minimum(int i, int j) " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*----------------------------------------------------------------
float minimum
----------------------------------------------------------------*/
static void minimum(double i, double j) {
  cout << " minimum(double i, double j " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*----------------------------------------------------------------
ch minimum
----------------------------------------------------------------*/
static void minimum(char i, char j) {
  cout << " minimum(char i, char j) " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*----------------------------------------------------------------
string minimum
----------------------------------------------------------------*/
static void minimum(const char* i, const char* j) {
  cout << " minimum(const char* i, const char* j) " << endl ;
```

```
  int x = strcmp(i,j) ;
  if (x < 0)
    cout << i << " is less than " << j << endl ;
  else if (x == 0) {
    cout << i << " is equal to " << j << endl ;
  }else {
    cout << i << " is greater than  " << j << endl ;
  }
}

/*-----------------------------------------------------------------
define minimum
-----------------------------------------------------------------*/
#define tminimum(i,j) { \
  cout << " tminimum(T i, T j) " << endl ; \
  if (i < j) { \
    cout << i << " is less than " << j << endl ; \
  }else { \
    cout << i << " is greater than or equal " << j << endl ; \
  } \
} \

/*-----------------------------------------------------------------
define print_array
-----------------------------------------------------------------*/
#define print_array(s, a, size) { \
  cout << s << " {"; \
  for (int i = 0; i < size; i++) { \
    cout << a[i] ; \
    if (i < size - 1) { \
      cout << ", " ; \
    } \
  } \
  cout << " } " << endl ; \
}\


/*-----------------------------------------------------------------
assignment
-----------------------------------------------------------------*/
static void assignment() {
  int a[] = { 1, 2, 3, 4, 5 };
  print_array("a is ",a,sizeof(a)/sizeof(int)) ;
  double b[] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
  print_array("b is ",b,sizeof(b)/sizeof(double)) ;
  char c[] = "HELLO";
  print_array("c is ",c,sizeof(c)/sizeof(char)) ;
}

```

```

/*-----------------------------------------------------------------
 minimum(int i, int j)
47 is less than 90
 minimum(double i, double j
47.9 is less than 90.8
 minimum(char i, char j)
a is less than b
 minimum(const char* i, const char* j)
jag is equal to jag
 minimum(const char* i, const char* j)
jaf is less than jag
 minimum(const char* i, const char* j)
jah is greater than  jag
 tminimum(T i, T j)
47 is less than 90
 tminimum(T i, T j)
47.9 is less than 90.8
 tminimum(T i, T j)
a is less than b
 tminimum(T i, T j)
jag is less than jag
 tminimum(T i, T j)
jaf is less than jag
 tminimum(T i, T j)
jah is less than jag
a is  {1, 2, 3, 4, 5 }
b is  {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 }
c is  {H, E, L, L, O,   }

-----------------------------------------------------------------*/
int main() {
  minimum(47,90) ;
  minimum(47.9,90.8) ;
  minimum('a','b') ;
  minimum("jag","jag") ;
  minimum("jaf","jag") ;
  minimum("jah","jag") ;
  tminimum(47,90) ;
  tminimum(47.9,90.8) ;
  tminimum('a','b') ;
  tminimum("jag","jag") ;
  tminimum("jaf","jag") ;
  tminimum("jah","jag") ;
  assignment();
  return 0 ;
}

```

10.3 Function templates

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: ftemp.cpp
----------------------------------------------------------------*/

#include <iostream>
using namespace std;

/*----------------------------------------------------------------
int minimum
----------------------------------------------------------------*/
static void minimum(int i, int j) {
  cout << " minimum(int i, int j) " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*----------------------------------------------------------------
float minimum
----------------------------------------------------------------*/
static void minimum(double i, double j) {
  cout << " minimum(double i, double j " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*----------------------------------------------------------------
ch minimum
----------------------------------------------------------------*/
static void minimum(char i, char j) {
  cout << " minimum(char i, char j) " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*----------------------------------------------------------------
string minimum
----------------------------------------------------------------*/
static void minimum(const char* i, const char* j) {
  cout << " minimum(const char* i, const char* j) " << endl ;
```

```
  int x = strcmp(i,j) ;
  if (x < 0)
    cout << i << " is less than " << j << endl ;
  else if (x == 0) {
    cout << i << " is equal to " << j << endl ;
  }else {
    cout << i << " is greater than  " << j << endl ;
  }
}

/*-----------------------------------------------------------------
template minimum
-----------------------------------------------------------------*/
template <typename T>
static void tminimum(T i, T j) {
  cout << " tminimum(T i, T j) " << endl ;
  if (i < j) {
    cout << i << " is less than " << j << endl ;
  }else {
    cout << i << " is greater than or equal " << j << endl ;
  }
}

/*-----------------------------------------------------------------
string minimum
-----------------------------------------------------------------*/
static void tminimum(const char* i, const char* j) {
  cout << " tminimum(const char* i, const char* j) " << endl ;
  int x = strcmp(i,j) ;
  if (x < 0)
    cout << i << " is less than " << j << endl ;
  else if (x == 0) {
    cout << i << " is equal to " << j << endl ;
  }else {
    cout << i << " is greater than  " << j << endl ;
  }
}

/*-----------------------------------------------------------------
template assignment
-----------------------------------------------------------------*/
template <typename T>
static void print_array(const char *s, const T& a, int size) {
  cout << s << " {";
  for (int i = 0; i < size; i++) {
    cout << a[i] ;
    if (i < size - 1) {
      cout << ", " ;
    }
```

```
  }
  cout << " } " << endl ;
}

/*-----------------------------------------------------------------
assignment
-----------------------------------------------------------------*/
static void assignment() {
  int a[] = { 1, 2, 3, 4, 5 };
  print_array("a is ",a,sizeof(a)/sizeof(int)) ;
  double b[] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
  print_array("b is ",b,sizeof(b)/sizeof(double)) ;
  char c[] = "HELLO";
  print_array("c is ",c,sizeof(c)/sizeof(char)) ;
}

/*-----------------------------------------------------------------
47.9 is less than 90.8
 minimum(char i, char j)
a is less than b
 minimum(const char* i, const char* j)
jag is equal to jag
 minimum(const char* i, const char* j)
jaf is less than jag
 minimum(const char* i, const char* j)
jah is greater than  jag
 tminimum(T i, T j)
47 is less than 90
 tminimum(T i, T j)
47.9 is less than 90.8
 tminimum(T i, T j)
a is less than b
 tminimum(const char* i, const char* j)
jag is equal to jag
 tminimum(const char* i, const char* j)
jaf is less than jag
 tminimum(const char* i, const char* j)
jah is greater than  jag
a is  {1, 2, 3, 4, 5 }
b is  {1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 }
c is  {H, E, L, L, O,   }
-----------------------------------------------------------------*/
int main() {
  minimum(47,90) ;
  minimum(47.9,90.8) ;
  minimum('a','b') ;
  minimum("jag","jag") ;
  minimum("jaf","jag") ;
  minimum("jah","jag") ;
```

```
148    tminimum(47,90) ;
149    tminimum(47.9,90.8) ;
150    tminimum('a','b') ;
151    tminimum("jag","jag") ;
152    tminimum("jaf","jag") ;
153    tminimum("jah","jag") ;
154    assignment();
155    return 0 ;
156  }
157
158
```

VASUDEVAMU . HV

10.4 Class templates

10.5 Integer stack implementation

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: istack.h
-----------------------------------------------------------------*/
#ifndef ISTACK_H
#define ISTACK_H

#include <iostream>
#include <cassert>
using namespace std;

/*-------------------------------------------------------
class istack
-------------------------------------------------------*/
namespace vasudevamurthy {
  class istack {
  public:
    static const int SIZE = 100;
    istack(int n = SIZE) ;
    ~istack() ;
    istack(const istack& s) ;
    istack& operator=(const istack& rhs) ;

    int size() const {return _sp; }
    int capacity() const {return _size ; }
    bool isempty() const {return _sp ? false : true ; }
    bool isfull() const {return (_sp < _size) ? false: true ;}
    void push(const int& x) ;
    int& top() const ;
    void pop() ;
    friend ostream& operator<<(ostream& o, const istack& s) ;
    static void set_verbose(bool x) {_display = x ; }

  private:
    int* _array ;
    int _size ;
    int _sp ;
    void _copy(const istack& s) ;
    static bool _display ;

  };
}

#endif
```

```
/*----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: istack.cpp
----------------------------------------------------------------*/

#include "istack.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*----------------------------------------------------------------
static function declaration
----------------------------------------------------------------*/
bool istack::_display = true ;

/*----------------------------------------------------------------
Constructor
----------------------------------------------------------------*/
istack::istack(int n):_size(n),_sp(0),_array(nullptr) {
  if (_display) {
    cout << "istack constructor" << endl ;
  }
  _array = new int[n] ;
}

/*----------------------------------------------------------------
destructor
----------------------------------------------------------------*/
istack::~istack() {
  if (_display) {
    cout << "istack destructor" << endl ;
  }
  delete [] _array ;
  _size = 0 ;
  _sp = 0 ;
}

/*----------------------------------------------------------------
_Copy
----------------------------------------------------------------*/
void istack::_copy(const istack& s){
  _size = s._size ;
  _sp = s._sp ;
  _array = new int[_size] ;
  for (int i = 0; i < _sp ; i++) {
    _array[i] = s._array[i] ;
  }
}

```

```
/*-----------------------------------------------------------------
Copy  Constructor
-----------------------------------------------------------------*/
istack::istack(const istack& s){
  if (_display) {
    cout << "istack Copy constructor" << endl ;
  }
  _copy(s) ;
}

/*-----------------------------------------------------------------
equal  Constructor
-----------------------------------------------------------------*/
istack& istack::operator=(const istack& s){
  if (_display) {
    cout << "istack equal operator" << endl ;
  }
  if (this != &s) {
    delete [] _array ;
    _copy(s) ;
  }
  return *this ;
}

/*-----------------------------------------------------------------
push
-----------------------------------------------------------------*/
void istack::push(const int& x){
  if (isfull()) {
    assert(0) ;
  }
  _array[_sp++] = x ;
}

/*-----------------------------------------------------------------
top
-----------------------------------------------------------------*/
int& istack::top() const{
  if (isempty()) {
    assert(0) ;
  }
  return _array[_sp-1] ;
}

/*-----------------------------------------------------------------
pop
-----------------------------------------------------------------*/
void istack::pop() {
  if (isempty()) {
```

```
    assert(0) ;
  }
  _sp-- ;
}

/*-----------------------------------------------------------------
Overloaded print operator
Because operator<< is global, we need to mention namespace
-----------------------------------------------------------------*/
namespace vasudevamurthy {
  ostream& operator<<(ostream& o, const istack& s) {
    for (int i = 0; i < s._sp; i++) {
      cout << s._array[i] << " " ;
    }
    return o ;
  }
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: istack_test.cpp

-----------------------------------------------------------------*/
#include "istack.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*-----------------------------------------------------------------
istack constructor
s1 looks like this: 0 100 200 300 400
istack Copy constructor
s2 looks like this: 0 100 200 300 400
s2 and s1 looks like this:
0 100 200 300 400
0 100 200 300 400
top of s1 = 400
After poping 4 elements s1 looks like this:
0
s2 looks like this:
0 100 200 300 400
istack equal operator
After s2 = s1, s2 looks like this:
0
istack destructor
istack destructor
-----------------------------------------------------------------*/
static void test() {
  istack s1 ;
  for (int i = 0; i < 5; i++) {
    s1.push(i * 100) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  istack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
```

```
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  istack::set_verbose(true) ;
  test() ;
  return 0 ;
}

```

VASUDEVAMURTHY

10.6 Generic stack implementation using typedef

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: tstack.h
-----------------------------------------------------------------*/
#ifndef tstack_H
#define tstack_H

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-----------------------------------------------------------------
Only place you have to change
-----------------------------------------------------------------*/
//#define TEST_INT
#define TEST_STRING
//#define TEST_UNSIGNED_LONG

/*-----------------------------------------------------------------
All typedef here
-----------------------------------------------------------------*/
#ifdef TEST_INT
typedef int T;
#endif

#ifdef TEST_STRING
typedef string T;
#endif

#ifdef TEST_UNSIGNED_LONG
typedef unsigned long T;
#endif

/*-------------------------------------------------------
class tstack
-------------------------------------------------------*/
namespace vasudevamurthy {
  class tstack {
  public:
    static const int SIZE = 100;
    tstack(int n = SIZE) ;
    ~tstack() ;
    tstack(const tstack& s) ;
    tstack& operator=(const tstack& rhs) ;

    int size() const {return _sp; }
    int capacity() const {return _size ; }
    bool isempty() const {return _sp ? false : true ; }
```

```
    bool isfull() const {return (_sp < SIZE) ? false: true ;}
    void push(const T& x) ;
    T& top() const ;
    void pop() ;
    friend ostream& operator<<(ostream& o, const tstack& s) ;
    static void set_verbose(bool x) {_display = x ; }

  private:
    T* _array ;
    int _size ;
    int _sp ;
    void _copy(const tstack& s) ;
    static bool _display ;
  };
}

#endif


```

VASUDEVA

```
/*----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: tstack.cpp
----------------------------------------------------------------*/

#include "tstack.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*----------------------------------------------------------------
static function declaration
----------------------------------------------------------------*/
bool tstack::_display = true ;

/*----------------------------------------------------------------
Constructor
----------------------------------------------------------------*/
tstack::tstack(int n):_size(n),_sp(0),_array(nullptr) {
  if (_display) {
    cout << "tstack constructor" << endl ;
  }
  _array = new T[n] ;
}

/*----------------------------------------------------------------
destructor
----------------------------------------------------------------*/
tstack::~tstack() {
  if (_display) {
    cout << "tstack destructor" << endl ;
  }
  delete [] _array ;
  _size = 0 ;
  _sp = 0 ;
}

/*----------------------------------------------------------------
_Copy
----------------------------------------------------------------*/
void tstack::_copy(const tstack& s){
  _size = s._size ;
  _sp = s._sp ;
  _array = new T[_size] ;
  for (int i = 0; i < _sp ; i++) {
    _array[i] = s._array[i] ;
  }
}

```

```
/*-----------------------------------------------------------------
Copy  Constructor
-----------------------------------------------------------------*/
tstack::tstack(const tstack& s){
  if (_display) {
    cout << "tstack Copy constructor" << endl ;
  }
  _copy(s) ;
}

/*-----------------------------------------------------------------
equal  Constructor
-----------------------------------------------------------------*/
tstack& tstack::operator=(const tstack& s){
  if (_display) {
    cout << "tstack equal operator" << endl ;
  }
  if (this != &s) {
    delete [] _array ;
    _copy(s) ;
  }
  return *this ;
}

/*-----------------------------------------------------------------
push
-----------------------------------------------------------------*/
void tstack::push(const T& x){
  if (isfull()) {
    assert(0) ;
  }
  _array[_sp++] = x ;
}

/*-----------------------------------------------------------------
top
-----------------------------------------------------------------*/
T& tstack::top() const{
  if (isempty()) {
    assert(0) ;
  }
  return _array[_sp-1] ;
}

/*-----------------------------------------------------------------
pop
-----------------------------------------------------------------*/
void tstack::pop() {
  if (isfull()) {
```

```
    assert(0) ;
  }
  _sp-- ;
}

/*-----------------------------------------------------------------
Overloaded print operator
Because operator<< is global, we need to mention namespace
-----------------------------------------------------------------*/
namespace vasudevamurthy {
  ostream& operator<<(ostream& o, const tstack& s) {
    for (int i = 0; i < s._sp; i++) {
      cout << s._array[i] << " " ;
    }
    return o ;
  }
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: tstack_main.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
-----------------------------------------------------------------*/
#include "tstack.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
#ifdef TEST_INT
static void test_int() {
  tstack s1 ;
  for (int i = 0; i < 5; i++) {
    s1.push(i * 100) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  tstack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}
#endif

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
#ifdef TEST_STRING
static void test_string() {
  tstack s1 ;
  for (int i = 0; i < 5; i++) {
    char a[2] ;
```

```
    a[0] = i + '0' ;
    a[1] = '\0' ;
    string s = "course " ;
    s1.push(s + a) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  tstack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}
#endif

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  tstack::set_verbose(true) ;
#ifdef TEST_INT
  test_int() ;
#endif
#ifdef TEST_STRING
  test_string() ;
#endif
  return 0 ;
}

```

10.7 Stack implementation using inheritence

10.7.1 Problem of using objects and static binding

```
class B {
public:
  B() {cout << "in B constructor\n";}
  B(const B& t) {cout << "in B copy constructor\n";}
  virtual ~B() {cout << "in B destructor\n";}
  virtual void who_am_i() const {cout << " I am B\n"; }
};


class D:public B {
public:
  D() {cout << "in D constructor\n";}
  ~D() {cout << "in D destructor\n";}
  D(const D& r) {cout << "in D copy constructor\n"; }
  virtual void who_am_i() const {cout << " I am D\n";}
};

static void test_object() {
  B b ;
  D d ;
  b.who_am_i() ;  I am B
  d.who_am_i() ;  I am D
  B c = d ;
  c.who_am_i() ;  I am B
}

static void test_value() {
  B b ;
  D d ;
  b.who_am_i() ;
  d.who_am_i() ;
  B* c = &d ;
  c->who_am_i() ; I am D
}
static void test_alias() {
  B b ;
  D d ;
  b.who_am_i() ;
  d.who_am_i() ;
  B& c = d ;
  c.who_am_i() ; I am D
}
```

Figure 10.1: Using instance as object, pointers and reference

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: test.cpp
----------------------------------------------------------------*/
#include <iostream>
using namespace std;


/*----------------------------------------------------------------
Base class
----------------------------------------------------------------*/
class B {
public:
  B() {cout << "in B constructor\n";}
  B(const B& t) {cout << "in B copy constructor\n";}
  B& operator=(const B& r) { cout << "in B EQUAL operator \n";}
  virtual ~B() {cout << "in B destructor\n";}
  virtual void who_am_i() const {cout << " I am B\n"; }
};

/*----------------------------------------------------------------
Derived class
----------------------------------------------------------------*/
class D:public B {
public:
  D() {cout << "in D constructor\n";}
  ~D() {cout << "in D destructor\n";}
  D(const D& r) {cout << "in D copy constructor\n"; }
  D& operator=(const D& r) { cout << "in D EQUAL operator \n"; return *this; }
  virtual void who_am_i() const {cout << " I am D\n";}
};

/*----------------------------------------------------------------
in B constructor
in B constructor
in D constructor
 I am B
 I am D
in B copy constructor
 I am B
in B destructor
in D destructor
in B destructor
in B destructor
in B constructor
in B constructor
in D constructor
----------------------------------------------------------------*/
static void test_object() {
  B b ;
  D d ;
  b.who_am_i() ;
  d.who_am_i() ;

  /* problem */
  B c = d ;
  c.who_am_i() ;
}

/*----------------------------------------------------------------
in B constructor
in B constructor
in D constructor
 I am B
 I am D
 I am D
```


VASUDEVAMURTHY

```
in D destructor
in B destructor
in B destructor
-----------------------------------------------------------------*/
static void test_value() {
  B b ;
  D d ;
  b.who_am_i() ;
  d.who_am_i() ;

  /* problem resolved */
  B* c = &d ;
  c->who_am_i() ;
}

/*-----------------------------------------------------------------
in B constructor
in B constructor
in D constructor
 I am B
 I am D
 I am D
in D destructor
in B destructor
in B destructor
-----------------------------------------------------------------*/
static void test_alias() {
  B b ;
  D d ;
  b.who_am_i() ;
  d.who_am_i() ;

  /* problem resolved */
  B& c = d ;
  c.who_am_i() ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  test_object() ;
  test_value() ;
  test_alias() ;
  return 0 ;
}

```

VASUDEVAMURTHY

Figure 10.2: Adding derived data by using base base pointer

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: vstack.h
----------------------------------------------------------------*/
#ifndef vstack_H
#define vstack_H

#include <iostream>
#include <cassert>
#include <string>
using namespace std;


/*----------------------------------------------------------------
DATA is T
----------------------------------------------------------------*/
class T {
public:
  T() {cout << "in T constructor\n";}
  T(const T& t) {cout << "in T copy constructor\n";}
  T& operator=(const T& r) { cout << "in T EQUAL operator \n";}
  virtual ~T() {cout << "in T destructor\n";}

  virtual T* clone() const = 0 ; //Uses the copy constructor
  virtual T* create() const = 0 ; //Uses the default constructor

  //friend ostream& operator<<(ostream& o, const T& r) { return o; }
  //Operator <<:  It's not a member function, so it can't be made virtual. However, it could
  //delegate to a virtual member function

  virtual ostream& print(ostream& o) const = 0 ; //PURE VIRTIAL FUNCTION - ABSTRACT CLASS
};


/*-------------------------------------------------------
class vstack
-------------------------------------------------------*/
namespace vasudevamurthy {
  class vstack {
  public:
    enum {SIZE = 100} ;
    vstack(int n = SIZE) ;
    ~vstack() ;
    vstack(const vstack& s) ;
    vstack& operator=(const vstack& rhs) ;

    int size() const {return _sp; }
    int capacity() const {return _size ; }
    bool isempty() const {return _sp ? false : true ; }
    bool isfull() const {return (_sp < SIZE) ? false: true ;}
    void push(T* x) ;
    T* top() const ;
    void pop() ;
    friend ostream& operator<<(ostream& o, const vstack& s) ;
    static void set_verbose(bool x) {_display = x ; }

  private:
    T** _array ;
    int _size ;
    int _sp ;
    void _copy(const vstack& s) ;
    static bool _display ;
    void _release() ;
  };
}

```

VASUDEVAMURTHY

Figure 10.3: operation top

VASUDEVAMURTHY

Figure 10.4: operation pop

VASUDEVAMURTHY

Figure 10.5: stack copy and equal operator

VASUDEVAMURTHY

Figure 10.6: stack destructor

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: vstack.cpp
----------------------------------------------------------------*/

#include "vstack.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*----------------------------------------------------------------
static function declaration
----------------------------------------------------------------*/
bool vstack::_display = true ;

/*----------------------------------------------------------------
Constructor
----------------------------------------------------------------*/
vstack::vstack(int n):_size(n),_sp(0),_array(NULL) {
  if (_display) {
    cout << "vstack constructor" << endl ;
  }
  _array = new T*[n] ;
}

/*----------------------------------------------------------------
release
----------------------------------------------------------------*/
void vstack::_release() {
  for (int i = 0; i < _sp; i++) {
    delete _array[i] ; /* virtual destructor called here */
  }
  delete [] _array ;
}

/*----------------------------------------------------------------
destructor
----------------------------------------------------------------*/
vstack::~vstack() {
  if (_display) {
    cout << "vstack destructor" << endl ;
  }
  _release() ;
  _size = 0 ;
  _sp = 0 ;
}

/*----------------------------------------------------------------
_Copy
----------------------------------------------------------------*/
void vstack::_copy(const vstack& s){
  _size = s._size ;
  _sp = s._sp ;
  _array = new T*[_size] ;
  for (int i = 0; i < _sp ; i++) {
    T* si = s._array[i] ;
    T* copyp = si->clone() ; /* virtual so that you copy derived obj */
    _array[i] = copyp ;
  }
}

/*----------------------------------------------------------------
Copy  Constructor
----------------------------------------------------------------*/
vstack::vstack(const vstack& s){
  if (_display) {
```


VASUDEVAMURTHY

```
    cout << "vstack Copy constructor" << endl ;
  }
  _copy(s) ;
}

/*-----------------------------------------------------------------
equal  Constructor
-----------------------------------------------------------------*/
vstack& vstack::operator=(const vstack& s){
  if (_display) {
    cout << "vstack equal operator" << endl ;
  }
  if (this != &s) {
    _release() ;
    _copy(s) ;
  }
  return *this ;
}

/*-----------------------------------------------------------------
push
-----------------------------------------------------------------*/
void vstack::push(T* x){
  if (isfull()) {
    assert(0) ;
  }
  //PROBLEM. We don't know that x is a pointer to an object
  //or x is a pointer to a pointer object
  T* copyx = x->clone() ; /* virtual so that you copy derived obj */
  _array[_sp++] = copyx ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
T* vstack::top() const{
  if (isempty()) {
    assert(0) ;
  }
  return _array[_sp-1] ;
}

/*-----------------------------------------------------------------
pop
-----------------------------------------------------------------*/
void vstack::pop() {
  if (isfull()) {
    assert(0) ;
  }
  T* si = _array[_sp-1] ;
  delete si ; /* virtual so that you free derived obj */
  --_sp ;
}

/*-----------------------------------------------------------------
Overloaded print operator
Because operator<< is global, we need to mention namespace
-----------------------------------------------------------------*/
namespace vasudevamurthy {
  ostream& operator<<(ostream& o, const vstack& s) {
    for (int i = 0; i < s._sp; i++) {
      T* x = s._array[i] ;
      //Note that x will be derived object pointer here
      x->print(o) ;
      o << " " ;
    }
```

VASUDEVAMURTHY

```
    return o ;
  }
}

```

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: vstack_main.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
Program exited with status code 0.
----------------------------------------------------------------*/
#include "vstack.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*----------------------------------------------------------------
Class Int
----------------------------------------------------------------*/
class Int:public T {
public:
  Int(int x=0): _x(x) {cout << "in Int constructor\n";}
  ~Int() {cout << "in Int destructor\n";}
  Int(const Int& r) {cout << "in Int copy constructor\n";_x = r._x ; }
  Int& operator=(const Int& r) { cout << "in Int EQUAL operator \n";_x = r._x ; return *this; }

  virtual Int* clone() const { return new Int(*this) ;} // You need a virtual destructor here
  virtual Int* create() const { return new Int(); }

  ostream& print(ostream& o) const { o << *this; return o ; }
  friend ostream& operator<<(ostream& o, const Int& r) { cout << r._x; return o; }
  const int& operator() () const {return _x; } //calls a function (), that returns int and takes nothing as input

private:
  int _x ;
};

/*----------------------------------------------------------------

----------------------------------------------------------------*/
static void test_int_by_value() {
  vstack s1 ;
  for (int i = 0; i < 5; i++) {
    Int t(i*100) ;
    s1.push(&t) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;

  vstack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  T* y = s1.top();
  //Int* z = y->clone() ;
  Int* pz = dynamic_cast<Int*> (y) ;
  assert(pz) ;
  Int& z = *(pz) ;
  cout << "top of s1 = " << z() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
```

```
  cout << s2 << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void test_int_by_address() {
  vstack s1 ;
  for (int i = 0; i < 5; i++) {
    Int* t = new Int(i*100) ;
    s1.push(t) ;
    delete t ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;

  vstack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  T* y = s1.top();
  //Int* z = y->clone() ;
  Int *pz = dynamic_cast<Int*> (y) ;
  assert(pz) ;
  Int& z = *(pz) ;
  cout << "top of s1 = " << z() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}

/*-----------------------------------------------------------------
Class String
-----------------------------------------------------------------*/
class String:public T {
public:
  String() {cout << "in String Default constructor\n";}
  String(string x): _x(x) {cout << "in String constructor\n";}
  ~String() {cout << "in String destructor\n";}
  String(const String& r) {cout << "in String copy constructor\n";_x = r._x ; }
  String& operator=(const String& r) { cout << "in String EQUAL operator \n";_x = r._x ; return *this;
   }

  virtual String* clone() const { return new String(*this) ;} // You need a virtual destructor here
  virtual String* create() const { return new String(); }

  ostream& print(ostream& o) const { o << *this; return o ; }
  friend ostream& operator<<(ostream& o, const String& r) { cout << r._x; return o; }
  const string& operator() () const {return _x; } //calls a function (), that returns int and takes
  nothing as input

private:
  string _x ;
};

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
```

```
static void test_string() {
  vstack s1 ;
  for (int i = 0; i < 5; i++) {
    char a[2] ;
    a[0] = i + '0' ;
    a[1] = '\0' ;
    string s = "course ";
    string sa = s + a ;
    String f(sa) ;
    s1.push(&f) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  vstack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;

  T* y = s1.top();
  //Int* z = y->clone() ;
  String *pz = dynamic_cast<String*> (y) ;
  assert(pz) ;
  String& z = *(pz) ;
  cout << "top of s1 = " << z() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}


/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  vstack::set_verbose(true) ;
  test_int_by_value() ;
  test_int_by_address();
  test_string() ;
  return 0 ;
}

```

VASUDEVAMURTHY

10.8 Stack implementation using template

Difference between templates and macros

1. Macro gets executed at compile time. Macros are expanded by the preprocessor, the compiler doesn't see them. They are simple test substitution. This is why macros can't be scoped by classes, namespaces, etc., because those haven't been parsed yet. The simplicity and flexibility of a macro is exactly what's called for some purposes.

2. There is no way for the compiler to verify that the macro parameters are of compatible types. The macro is expanded without any special type checking.

3. Template gets executed at run time. A template is a way to make functions independent of data-types. This cannot be accomplished using macros. E.g. a sorting function doesn't have to care whether it's sorting integers or letters since the same algorithm might apply anyway.

4. Template is known to the compiler and hence provides stronger type checking. A template is defined to expand as if it were text, so the compiler can then compile the result. Naturally this is more type safe than a macro.

5. Template is also being available for classes, but macros are just functions.

Figure 10.7: Templates versus macros

VASUDEVAMURTHY

Inheritance VERSUS template

1.The inheritance approach is simply a 2-step process where in template it is a one-step process.

2. Inheritance first assigns something to a "type". The behavior is then "inherited" in a way by being a member of that type.

3.One disadvantage of being type-tied is that if typing {is-a} turns out to be the wrong modeling approach, then you have a lot of changing to do.

Figure 10.8: Inheitance versus template

Template has both declaration and implementation in one file. Why?

1.Template gets executed at run time.

That is the reason why, no code is generated when the compiler sees the templatized function bodies.

2. Code is generated only when we declare an object with a specific template argument. For this reason, declaration and implementation must be in the header files.

3. To create the code, C++ needs to know the function definitions. Since C++ typically creates functions for a template class when it sees a declaration of an instance of that class, it must know the function definitions at that time. Thus, they must appear in the *.h file.

Figure 10.9: Why all in one file?

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: stack.h
-----------------------------------------------------------------*/
#ifndef stack_H
#define stack_H

#include <iostream>
#include <cassert>
#include <string>
using namespace std;

/*-------------------------------------------------------
class stack
-------------------------------------------------------*/
namespace vasudevamurthy {
  template <typename T>
  class stack {
  public:
    static const int SIZE = 100 ;
    stack(int n = SIZE) ;
    ~stack() ;
    stack(const stack<T>& s) ;
    stack<T>& operator=(const stack<T>& rhs) ;

    int size() const {return _sp; }
    int capacity() const {return _size ; }
    bool isempty() const {return _sp ? false : true ; }
    bool isfull() const {return (_sp < SIZE) ? false: true ;}
    void push(const T& x) ;
    T& top() const ;
    void pop() ;
    //Note the call below
    template <typename U> friend ostream& operator<<(ostream& o, const stack<U>& s) ;
    static void set_verbose(bool x) {_display = x ; }

  private:
    T* _array ;
    int _size ;
    int _sp ;
    void _copy(const stack<T>& s) ;
    static bool _display ;
  };
}

/*-------------------------------------------------------
template implementations must be seen by compiler
-------------------------------------------------------*/
```

```
#include "stack_impl.h"

#endif


```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: stack_impl.h
-----------------------------------------------------------------*/
using namespace vasudevamurthy ;

/*-----------------------------------------------------------------
static function declaration
-----------------------------------------------------------------*/
template <typename T>
bool stack<T>::_display = true ;

/*-----------------------------------------------------------------
Constructor
-----------------------------------------------------------------*/
template <typename T>
stack<T>::stack(int n):_size(n),_sp(0),_array(nullptr) {
  if (_display) {
    cout << "stack constructor" << endl ;
  }
  _array = new T[n] ;
}

/*-----------------------------------------------------------------
destructor
-----------------------------------------------------------------*/
template <typename T>
stack<T>::~stack() {
  if (_display) {
    cout << "stack destructor" << endl ;
  }
  delete [] _array ;
  _size = 0 ;
  _sp = 0 ;
}

/*-----------------------------------------------------------------
_Copy
-----------------------------------------------------------------*/
template <typename T>
void stack<T>::_copy(const stack<T>& s){
  _size = s._size ;
  _sp = s._sp ;
  _array = new T[_size] ;
  for (int i = 0; i < _sp ; i++) {
    _array[i] = s._array[i] ;
  }
}

```

```
/*-----------------------------------------------------------------
Copy  Constructor
-----------------------------------------------------------------*/
template <typename T>
stack<T>::stack(const stack<T>& s){
  if (_display) {
    cout << "stack Copy constructor" << endl ;
  }
  _copy(s) ;
}

/*-----------------------------------------------------------------
equal  Constructor
-----------------------------------------------------------------*/
template <typename T>
stack<T>& stack<T>::operator=(const stack<T>& s){
  if (_display) {
    cout << "stack equal operator" << endl ;
  }
  if (this != &s) {
    delete [] _array ;
    _copy(s) ;
  }
  return *this ;
}

/*-----------------------------------------------------------------
push
-----------------------------------------------------------------*/
template <typename T>
void stack<T>::push(const T& x){
  if (isfull()) {
    assert(0) ;
  }
  _array[_sp++] = x ;
}

/*-----------------------------------------------------------------
top
-----------------------------------------------------------------*/
template <typename T>
T& stack<T>::top() const{
  if (isempty()) {
    assert(0) ;
  }
  return _array[_sp-1] ;
}

/*-----------------------------------------------------------------
```

```
pop
-----------------------------------------------------------------*/
template <typename T>
void stack<T>::pop() {
  if (isfull()) {
    assert(0) ;
  }
  _sp-- ;
}

/*-----------------------------------------------------------------
Overloaded print operator
Because operator<< is global, we need to mention namespace
-----------------------------------------------------------------*/
namespace vasudevamurthy {
  template <typename T>
  ostream& operator<<(ostream& o, const stack<T>& s) {
    for (int i = 0; i < s._sp; i++) {
      cout << s._array[i] << " " ;
    }
    return o ;
  }
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2016 Author: Jagadeesh Vasudevamurthy
Filename: Int.cpp stack_test.cpp

Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
-----------------------------------------------------------------*/
#include "stack.h"
#include "Int.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*-----------------------------------------------------------------
stack constructor
s1 looks like this: 0 100 200 300 400
stack Copy constructor
s2 looks like this: 0 100 200 300 400
s2 and s1 looks like this:
0 100 200 300 400
0 100 200 300 400
top of s1 = 400
After poping 4 elements s1 looks like this:
0
s2 looks like this:
0 100 200 300 400
stack equal operator
After s2 = s1, s2 looks like this:
0
stack destructor
stack destructor
-----------------------------------------------------------------*/
static void test_int_by_value() {
  stack<int> s1 ;
  for (int i = 0; i < 5; i++) {
    s1.push(i * 100) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  stack<int> s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
```

```
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}

/*-----------------------------------------------------------------
stack constructor
s1 looks like this: 003B74F0 003B7530 003B7570 003B75B0 003B75F0
stack Copy constructor
s2 looks like this: 003B74F0 003B7530 003B7570 003B75B0 003B75F0
s2 and s1 looks like this:
003B74F0 003B7530 003B7570 003B75B0 003B75F0
003B74F0 003B7530 003B7570 003B75B0 003B75F0
top of s1 = 400
stack destructor
stack destructor
-----------------------------------------------------------------*/
static void test_int_by_address() {
  stack<int*> s1 ;
  for (int i = 0; i < 5; i++) {
    int* t = new int(i*100) ;
    s1.push(t) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;

  stack<int*> s2(s1) ; //Now s2 and s1 will be holding same pointers
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  int*& y = s1.top();
  cout << "top of s1 = " << *(y) << endl ;
  for (int i = 0 ; i < 4; i++) {
    int*& y = s1.top();
    delete y ;
    s1.pop() ;
  }

  /* s2 is holding some pointers which is removed by s1 */
  /* DO NOT do d2 = s1 */

  while(!(s1.isempty())) {
    int*& y = s1.top();
    delete y ;
    s1.pop() ;
  }
```

```
  /* Why we should NOT do for s2 */
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void test_Int_by_value() {
  stack<Int> s1 ;
  for (int i = 0; i < 5; i++) {
    s1.push(i * 100) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  stack<Int> s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void test_Int_by_address() {
  stack<Int*> s1 ;
  for (int i = 0; i < 5; i++) {
    Int* t = new Int(i*100) ;
    s1.push(t) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;

  stack<Int*> s2(s1) ; //Now s2 and s1 will be holding same pointers
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  Int*& y = s1.top();
```

```
  cout << "top of s1 = " << *(y) << endl ;
  for (int i = 0 ; i < 4; i++) {
    Int*& y = s1.top();
    delete y ;
    s1.pop() ;
  }

  /* s2 is holding some pointers which is removed by s1 */
  /* DO NOT do d2 = s1 */

  while(!(s1.isempty())) {
    Int*& y = s1.top();
    delete y ;
    s1.pop() ;
  }
  /* Why we should NOT do for s2 */
}

/*-----------------------------------------------------------------
stack constructor
s1 looks like this: course 0 course 1 course 2 course 3 course 4
stack Copy constructor
s2 looks like this: course 0 course 1 course 2 course 3 course 4
s2 and s1 looks like this:
course 0 course 1 course 2 course 3 course 4
course 0 course 1 course 2 course 3 course 4
top of s1 = course 4
After poping 4 elements s1 looks like this:
course 0
s2 looks like this:
course 0 course 1 course 2 course 3 course 4
stack equal operator
After s2 = s1, s2 looks like this:
course 0
stack destructor
stack destructor
-----------------------------------------------------------------*/
static void test_string() {
  stack<string> s1 ;
  for (int i = 0; i < 5; i++) {
    char a[2] ;
    a[0] = i + '0' ;
    a[1] = '\0' ;
    string s = "course " ;
    s1.push(s + a) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  stack<string> s2(s1) ;
```

```
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}


/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
  stack<int>::set_verbose(true) ; //Note this call
  test_int_by_value() ;
  test_int_by_address();
  test_Int_by_value() ;
  test_Int_by_address();
  test_string();
  return 0 ;
}

```

VASUL . HV

VASUDEVAMURTHY

Chapter 11

Exception handling

11.1 Introduction

11.2 Problem

VASUDEVAMURTHY

What is the problem?

```
class data {
public:
  data(int x, int y):_x(x),_y(y) {
     cout << "In data constructor of " << _x << "," << _y << "\n" ;
  }
  ~data() {
    cout << "In data destructor of " << _x << "," << _y << "\n" ;
   }
  int _x ;
  int _y ;
};

static double divide(const data& d) {
  if (d._y == 0) {
    exit(0) ; //I cannot return valid double here */
  }
  return (double(d._x) / double(d._y)) ;
}

static void test() {
  data d1(3,2) ;
  cout << d1._x << "/" << d1._y << "= " << divide(d1) << endl ;
  data d2(3,0) ;
  cout << d2._x << "/" << d2._y << divide(d2) << endl ;
  data d3(3,8) ;
  cout << d3._x << "/" << d3._y << divide(d3) << endl ;
  /* d1 and d2 destructors are NOT called */
}
```

MEMORY LEAK

```
In data constructor of 3,2
3/2= 1.5
In data constructor of 3,0
```

Figure 11.1: Cannot handle unexpected input

VASUDEVAMURTHY

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: p1.cpp
-----------------------------------------------------------------*/

#include <iostream>
#include <iostream>
#include <cassert>
using namespace std;

/*-----------------------------------------------------------------
mydata
-----------------------------------------------------------------*/
class mydata {
public:
  mydata(int x, int y):_x(x),_y(y) { cout << "In mydata constructor of " << _x << "," << _y << "\n" ;}
  ~mydata() { cout << "In mydata destructor of " << _x << "," << _y << "\n" ;}
  int _x ;
  int _y ;
};

/*-----------------------------------------------------------------
divide
-----------------------------------------------------------------*/
static double divide(const mydata& d) {
  if (d._y == 0) {
    exit(0) ; //I cannot return valid double here */
  }
  return (double(d._x) / double(d._y)) ;
}

/*-----------------------------------------------------------------
In mydata constructor of 3,2
3/2= 1.5
In mydata constructor of 3,0
-----------------------------------------------------------------*/
static void test() {
  mydata d1(3,2) ;
  cout << d1._x << "/" << d1._y << "= " << divide(d1) << endl ;
  mydata d2(3,0) ;
  cout << d2._x << "/" << d2._y << divide(d2) << endl ;
  mydata d3(3,8) ;
  cout << d3._x << "/" << d3._y << divide(d3) << endl ;
  /* d1 and d2 destructors are NOT called */
}

/*-----------------------------------------------------------------
main
```

```
49 ----------------------------------------------------------------*/
50 int main() {
51   test() ;
52   return 0 ;
53 }
```

VASUDEVAMURTHY

11.3 Try and catch

VASUDEVAMURTHY

TRY and CATCH

```
class data {
public:
  data(int x, int y):_x(x),_y(y) {
     cout << "In data constructor of " << _x << "," << _y << "\n" ;
  }
  ~data() {
    cout << "In data destructor of " << _x << "," << _y << "\n" ;
   }
  int _x ;
  int _y ;
};
```

```
static double divide(const data& d) throw(const char*){
  if (d._y == 0) {
    throw "In divide: DIVISION BY ZERO" ;
    //I cannot return valid double here */
  }
  return (double(d._x) / double(d._y)) ;
}

static void test() {
  data d1(3,2) ;
  cout << d1._x << "/" << d1._y << "= " << divide(d1) << endl ;
  data d2(3,0) ;
  cout << d2._x << "/" << d2._y << divide(d2) << endl ;
  data d3(3,8) ;
  cout << d3._x << "/" << d3._y << divide(d3) << endl ;
  /* d1 and d2 destructors are called */
}

static void try_test() {
  try {
    test() ;
  }
  catch(const char* s) {
    cout << "I am in try_test: " << s << endl ;
  }
}
```

NO MEMORY LEAK

```
In data constructor of 3,2
3/2= 1.5
In data constructor of 3,0
In data destructor of 3,0
In data destructor of 3,2
I am in try_test: In divide: DIVI
```

SEPARATING TRY AND CATCH

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Figure 11.2: No more memory leak

VASUDEVAMURTHY

Why C setjmp and longjmp cannot be used in C++

One of the important aspects of C++ exception handing is: Unwinding the stack.

It is certainly possible that just prior to an exception being thrown, one or more user-defined types have already been instantiated. Furthermore, let's assume that the constructor functions for these types have allocated private resources on the heap. So when the exception is thrown, what happens to these resources as the instances go out of scope?

The answer is that the stack is automatically unwound and the destructor function is guaranteed to be called, thereby negating the possibility of memory leakage.

In C there is no concept of automatic invoking of destructors. Also C library functions setjmp and longjmp has no idea of calling destructors. This makes setjmp and longjmp unsatisfactory in C++. The C techniques of signals and setjmp/longjmp do not call destructors, so objects aren't properly cleaned up. This makes it virtually impossible to effectively recover from an exceptional condition because you'll always leave objects behind that haven't been cleaned up and that can no longer be accessed

Can you have a catch block with no preceding try block?

No. Note that an exception handler, also known as a catch block, can appear only after a try block or another exception handler. There can be no intervening statements. The process of throwing and catching an exception entails searching the exception handlers after the try block for the first one that can be invoked.

Figure 11.3: Need for try and catch

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: p2.cpp
-----------------------------------------------------------------*/

#include <iostream>
#include <iostream>
#include <cassert>
using namespace std;

/*-----------------------------------------------------------------
mydata
-----------------------------------------------------------------*/
class mydata {
public:
  mydata(int x, int y):_x(x),_y(y) { cout << "In mydata constructor of " << _x << "," << _y << "\n" ;}
  ~mydata() { cout << "In mydata destructor of " << _x << "," << _y << "\n" ;}
  int _x ;
  int _y ;
};

/*-----------------------------------------------------------------
divide
-----------------------------------------------------------------*/
static double divide(const mydata& d) throw(const char*){
  if (d._y == 0) {
    throw "In divide: DIVISION BY ZERO" ;
    //I cannot return valid double here */
  }
  return (double(d._x) / double(d._y)) ;
}

/*-----------------------------------------------------------------
In mydata constructor of 3,2
3/2= 1.5
In mydata constructor of 3,0
In mydata destructor of 3,0
In mydata destructor of 3,2
I am in try_test: In divide: DIVISION BY ZERO
-----------------------------------------------------------------*/
static void test() {
  mydata d1(3,2) ;
  cout << d1._x << "/" << d1._y << "= " << divide(d1) << endl ;
  mydata d2(3,0) ;
  cout << d2._x << "/" << d2._y << divide(d2) << endl ;
  mydata d3(3,8) ;
  cout << d3._x << "/" << d3._y << divide(d3) << endl ;
  /* d1 and d2 destructors are called */
```

```
}

/*-----------------------------------------------------------------
Separting try and catch
-----------------------------------------------------------------*/
static void try_test() {
  try {
    test() ;
  }
  catch(const char* s) {
    cout << "I am in try_test: " << s << endl ;
  }
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  try_test() ;
  return 0 ;
}
```

VASUDEVA

11.4 Stack example

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: error.h
-----------------------------------------------------------------*/
#ifndef ERROR_H
#define ERROR_H

#include <iostream>
using namespace std;

#ifdef _MSC_VER
#pragma warning(disable: 4996) /* Disable deprecation */
#endif

/*-------------------------------------------------------
class error
-------------------------------------------------------*/
namespace vasudevamurthy {
  class error{
  public:
    error(const char *name = NULL, const int num = -1): _name(NULL),_num(num) {
      if (name) {
        cout << "In error constructor\n" ;
        _name = new char[strlen(name) + 1] ;
        strcpy(_name,name) ;
      }
    }
    virtual ~error() {
      cout << "In error destructor\n" ;
      delete [] _name ;
    }

    error(const error& r) {
      cout << "In error copy constructor\n" ;
      _copy(r) ;
    }

    error& operator=(const error& r) {
      cout << "In error Equal constructor\n" ;
      if (this != &r) {
        delete [] _name ;
        _copy(r) ;
      }
      return *this ;
    }

    const char* name() const {return _name ;}
    int num() const {return _num; }
    virtual void print(ostream& o, char const *s) const{
```

```
      o << s << "Name = " << _name  << " Index = " << _num << endl ;
    }

  private:
    char *_name ;
    int _num ;

    void _copy(const error& r) {
      int l = strlen(r._name + 1) ;
      _name = new char[l] ;
      strcpy(_name,r._name) ;
      _num = r._num ;
    }
  };

  /*-------------------------------------------------------
  Underflow errors when poping an empty stack happens.
  This problem is handled here
  -------------------------------------------------------*/
  class error_underflow: public error{
  public:
    error_underflow(const char *name = NULL, const int num = -1): error
     (name,num) {
      cout << "In error_underflow constructor\n" ;
    }

    virtual ~error_underflow() {
      cout << "In error_underflow destructor\n" ;
    }

    error_underflow(const error_underflow& r):error(r) {
      cout << "In error_underflow copy constructor\n" ;
    }

    error_underflow& operator=(const error_underflow& r) {
      cout << "In error_underflow Equal constructor\n" ;
      if (this != &r) {
        error::operator=(r) ;
      }
      return *this ;
    }

    virtual void print(ostream& o, char const *s) const {
      o << s ;
      error::print(o,"UNDERFLOW Condition ") ;
    }
  };

  /*-------------------------------------------------------
```

```
Overflow errors when pushing elements to a stack that is already full.
This problem is handled here
-----------------------------------------------------------*/
class error_overflow: public error{
public:
  error_overflow(const char *name = NULL, const int num = -1): error (name,num) {
    cout << "In error_overflow constructor\n" ;
  }

  virtual ~error_overflow() {
    cout << "In error_overflow destructor\n" ;
  }

  error_overflow(const error_overflow& r):error(r) {
    cout << "In error_overflow copy constructor\n" ;
  }

  error_overflow& operator=(const error_overflow& r) {
    cout << "In error_overflow Equal constructor\n" ;
    if (this != &r) {
      error::operator=(r) ;
    }
    return *this ;
  }

  virtual void print(ostream& o, char const *s) const {
    o << s ;
    error::print(o,"OVERFLOW Condition ") ;
  }
};

/*-------------------------------------------------------
Out of memory heap space is handled here
-----------------------------------------------------------*/
class error_out_of_memory: public error{
public:
  error_out_of_memory(const char *name = NULL, const int num = -1): error (name,num) {
    cout << "In error_out_of_memory constructor\n" ;
  }

  virtual ~error_out_of_memory() {
    cout << "In error_out_of_memory destructor\n" ;
  }

  error_out_of_memory(const error_out_of_memory& r):error(r) {
    cout << "In error_out_of_memory copy constructor\n" ;
  }
```

```
145
146     error_out_of_memory& operator=(const error_out_of_memory& r) {
147       cout << "In error_out_of_memory Equal constructor\n" ;
148       if (this != &r) {
149         error::operator=(r) ;
150       }
151       return *this ;
152     }
153
154     virtual void print(ostream& o, char const *s) const {
155       o << s ;
156       error::print(o,"OUT_OF_MEMORY Condition ") ;
157     }
158   };
159 }
160
161 #endif
162
```

VASUDEVAN

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: istack.h
-----------------------------------------------------------------*/
#ifndef ISTACK_H
#define ISTACK_H

#include <iostream>
#include <cassert>
using namespace std;

/*-------------------------------------------------------
class istack
-------------------------------------------------------*/
namespace vasudevamurthy {
  class istack {
  public:
    enum {SIZE = 100} ;
    istack(int n = SIZE) ;
    ~istack() ;
    istack(const istack& s) ;
    istack& operator=(const istack& rhs) ;

    int size() const {return _sp; }
    int capacity() const {return _size ; }
    bool isempty() const {return _sp ? false : true ; }
    bool isfull() const {return (_sp < _size) ? false: true ;}
    void push(const int& x) ;
    int& top() const ;
    void pop() ;
    friend ostream& operator<<(ostream& o, const istack& s) ;
    static void set_verbose(bool x) {_display = x ; }

  private:
    int* _array ;
    int _size ;
    int _sp ;
    void _copy(const istack& s) ;
    static bool _display ;

  };
}

#endif
```

```
/*----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: istack.cpp
----------------------------------------------------------------*/

#include "istack.h"
#include "error.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*----------------------------------------------------------------
static function declaration
----------------------------------------------------------------*/
bool istack::_display = true ;

/*----------------------------------------------------------------
Constructor
----------------------------------------------------------------*/
istack::istack(int n):_size(n),_sp(0),_array(NULL) {
  if (_display) {
    cout << "istack constructor" << endl ;
  }
  _array = new int[n] ;
  if (!_array) {
    throw error_out_of_memory("istack",n) ;
  }
}

/*----------------------------------------------------------------
destructor
----------------------------------------------------------------*/
istack::~istack() {
  if (_display) {
    cout << "istack destructor" << endl ;
  }
  delete [] _array ;
  _size = 0 ;
  _sp = 0 ;
}

/*----------------------------------------------------------------
_Copy
----------------------------------------------------------------*/
void istack::_copy(const istack& s){
  _size = s._size ;
  _sp = s._sp ;
  _array = new int[_size] ;
  for (int i = 0; i < _sp ; i++) {
```

```
    _array[i] = s._array[i] ;
  }
}

/*-----------------------------------------------------------------
Copy  Constructor
-----------------------------------------------------------------*/
istack::istack(const istack& s){
  if (_display) {
    cout << "istack Copy constructor" << endl ;
  }
  _copy(s) ;
}

/*-----------------------------------------------------------------
equal  Constructor
-----------------------------------------------------------------*/
istack& istack::operator=(const istack& s){
  if (_display) {
    cout << "istack equal operator" << endl ;
  }
  if (this != &s) {
    delete [] _array ;
    _copy(s) ;
  }
  return *this ;
}

/*-----------------------------------------------------------------
push
-----------------------------------------------------------------*/
void istack::push(const int& x){
  if (isfull()) {
    throw error_overflow("istack",_sp) ;
  }
  _array[_sp++] = x ;
}

/*-----------------------------------------------------------------
top
-----------------------------------------------------------------*/
int& istack::top() const{
  if (isempty()) {
    throw error_underflow("istack",_sp) ;
  }
  return _array[_sp-1] ;
}

/*-----------------------------------------------------------------
```

```
pop
----------------------------------------------------------------*/
void istack::pop() {
  if (isfull()) {
    throw error_overflow("istack",_sp) ;
  }
  _sp-- ;
}

/*----------------------------------------------------------------
Overloaded print operator
Because operator<< is global, we need to mention namespace
----------------------------------------------------------------*/
namespace vasudevamurthy {
  ostream& operator<<(ostream& o, const istack& s) {
    for (int i = 0; i < s._sp; i++) {
      cout << s._array[i] << " " ;
    }
    return o ;
  }
}

```

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2011 Author: Jagadeesh Vasudevamurthy
Filename: istack.cpp istack_test.cpp

-----------------------------------------------------------------*/
#include "istack.h"
#include "error.h"
#include <iostream>
using namespace std;
using namespace vasudevamurthy ;

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void test() {
  istack s1(3) ;
  for (int i = 0; i < 5; i++) {
    s1.push(i * 100) ;
  }
  cout << "s1 looks like this: " ;
  cout << s1 << endl ;
  istack s2(s1) ;
  cout << "s2 looks like this: " ;
  cout << s2 << endl ;
  cout << "s2 and s1 looks like this: " << endl ;
  cout << s2 << endl << s1 << endl ;
  cout << "top of s1 = " << s1.top() << endl ;
  for (int i = 0 ; i < 4; i++) {
    s1.pop() ;
  }
  cout << "After poping 4 elements s1 looks like this: " << endl ;
  cout << s1 << endl ;
  cout << "s2 looks like this: " << endl ;
  cout << s2 << endl ;
  s2 = s1 ;
  cout << "After s2 = s1, s2 looks like this: " << endl ;
  cout << s2 << endl ;
}

/*-----------------------------------------------------------------
istack constructor
In error constructor
In error_overflow constructor
istack destructor <------------------- delete [] _array ;
test_exceptionOVERFLOW Condition Name = istack Index = 3
In error_overflow destructor <----
In error destructor <<--- delete [] _name  of error
-----------------------------------------------------------------*/
static void test_exception() {
```

```
50    try {
51      test() ;
52    }
53    catch (const error_underflow& a) {
54      a.print(cout,"test_exception") ;
55    }
56    catch(const error_overflow& a) {
57      a.print(cout,"test_exception") ;
58    }
59    catch(const error_out_of_memory& a) {
60      a.print(cout,"test_exception") ;
61    }
62    catch(...) {
63      cout <<  "How it is possible test_exception\n" ;
64    }
65  }
66
67  /*-----------------------------------------------------------------
68
69  -----------------------------------------------------------------*/
70  int main() {
71    istack::set_verbose(true) ;
72    test_exception();
73    return 0 ;
74  }
75
```

Chapter 12

Standard Template Library

12.1 Introduction

12.2 util class

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: util.h
----------------------------------------------------------------*/

/*----------------------------------------------------------------
include this file for all the programs
----------------------------------------------------------------*/
#ifndef UTIL_H
#define UTIL_H

/*----------------------------------------------------------------
Basic include files
----------------------------------------------------------------*/
#include <iostream>
#include <fstream>

#include <iomanip>      // std::setprecision
using namespace std;

#ifdef _WIN32
#include <cassert>
#include <ctime>
#else
#include <assert.h>
#include <time.h>
#include <string.h> //for strlen,strcat and strcpy on linux
#endif

//'sprintf' : This function or variable may be unsafe.Consider using sprintf_s instead.To disable deprecation,
//use _CRT_SECURE_NO_WARNINGS
//To overcome above warning
#ifdef _MSC_VER
#pragma warning(disable: 4996) /* Disable deprecation */
#endif


/*----------------------------------------------------------------
STL
----------------------------------------------------------------*/
#include <stdexcept> //Without this catch will NOT work on Linux
#include <vector>
#include <string>
#include <unordered_set>
#include <set>
#include <map>
#include <queue>
#include <algorithm>
```

VASUDEVAMURTHY

```
49
50 /*-------------------------------------------------------
51 class random number generator
52 -------------------------------------------------------*/
53 class Random {
54 public:
55   Random() { srand((unsigned)time(0)); }
56   //Gets a random from including a to including b
57   unsigned int get_random_number(unsigned int a = 0, unsigned int b = 10000)
      const {
58     unsigned int upper_bound, lower_bound;
59     if (a < b) { upper_bound = b - a; lower_bound = a; } else if (a >= b)
        { upper_bound = a - b; lower_bound = b; }
60
61     return(lower_bound + rand() % (upper_bound + 1));
62   }
63   /* no body can copy random or equal random */
64   Random(const Random& x) = delete;
65   Random& operator=(const Random& x) = delete;
66 private:
67
68 };
69
70 /*-----------------------------------------------------------------
71 functions for containers
72 -----------------------------------------------------------------*/
73 struct ascendingint {
74   bool operator() (const int& a, const int& b) {
75     return (a < b) ;
76   }
77 };
78
79 struct descendingint {
80   bool operator() (const int& a, const int& b) {
81     return (a > b) ;
82   }
83 };
84
85 struct printint {
86   void operator() (const int& a) {
87     cout << a << " " ;
88   }
89 };
90
91 struct add3int {
92   void operator() (int& a) {
93     a = a + 3 ;
94   }
95 };
```

```
 96
 97 struct displaySignint {
 98   void operator() (const int& a) {
 99     if (a > 0) {
100       std::cout << "+" ;
101     }
102     std::cout << a << " " ;
103   }
104 };
105 #endif
106
107 //EOF
108
```

VASUDEVAMURTHY

12.3 Concept of l-value and r-value

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: lvaluervalue.cpp

On linux:
g++ ../util/util.cpp lvaluervalue.cpp
valgrind a.out
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file teaches lvalue  and rvalue  concepts
-----------------------------------------------------------------*/
#include "../util/util.h"

/*-----------------------------------------------------------------
Some local function
-----------------------------------------------------------------*/
static int sum(int x, int y) {
  return (x + y);
}

static int square(int& x) {
  return (x*x);
}

/*-----------------------------------------------------------------
l-value: An object that occupies some identifiable location in memory
r-value: An object that is NOT a lvalue
-----------------------------------------------------------------*/
static void whatisLvalueAndRvalue() {
  class dog {
  public:
    dog() = default;
  private:
    int age_ = 0;
  };
  {
    int i; //i is a l-value because I can get it address. see line below
    int*p = &i; //i address is identifiable
    i = 2; //Content of that address is also modifiable
  }
  {
    dog d; //d is a l-value
  }
  {
    //Most of the C++ variables are l-value
  }

  {
```

```
    int x = 2; //2 is an r-value. Why? See below
    //2 = x ; //error C2106: '=': left operand must be l-value
    int y = x + 2; //x + 2 is an r-value. Why? because you cannot take it address. see below
    //int* p = &(x +2) ; //error C2102: '&' requires l-value
    //x + 2 = 4 ; //error C2106: '=': left operand must be l-value
    dog d1 = dog(); //dog() is r-value of user defined type
  }

  {
    //r-value functions
    int i = sum(4, 5); //sum(4,5) is r-value
    //sum(4,50) = 35 ; //error C2106: '=': left operand must be l-value
  }

  {
    //r-value reference
    int i = 5;
    int& j = i; //j is an l-value reference because i is an l-value
    //You cannot have reference to r-value.
    //int& r = 5 ; //error C2440: 'initializing': cannot convert from 'int' to 'int &'

    //One exception
    const int& r = 5;
    //Above is a two-step process for compiler
    const int temp = 5;
    const int& r1 = temp;
  }

  {
    //r-value function
    int i = 9;
    square(i); //Square is a function that takes l-value reference and returns l-value reference
    //square(9) ; //Square cannot take r-value reference.
    //error C2664: 'int square(int &)': cannot convert argument 1 from 'int' to 'int &'
  }


  {
    //l-value can be converted r-value
    int i = 1; //i is l-lvalue 1 is r-value
    int x = i + 2; //Now i + 2 is r-value
    int y = i; //i is an l-value. But in this context y = i, i is r-value.
    //l-value i is implicitely converted to r-value
  }

```

```
  {
    //r-value cannot be converted to l-value implicitely
    //r-value can be created explicitely
    int v[3];
    *(v + 2) = 4;
    //step1 : (v+2) is r-value
    //step2; *(v+2) is l-value. That's why we can assign 4 to l-value
  }
}

/*-----------------------------------------------------------------
Some local function
-----------------------------------------------------------------*/
//You cannot overload p with l-value reference or r-value reference
static void p(int x) {
  std::cout << "In static void p(int x)" << x << std::endl;
}
//You can overload only l-value reference or r-value reference
static void p1(int& x) {
  std::cout << "In static void p1(int& x)" << x << std::endl;
}

static void p1(int&& x) {
  std::cout << "In static void p1(int&& x)" << x << std::endl;
}


/*-----------------------------------------------------------------
l-value: An object that occupies some identifiable location in memory
r-value: An object that is NOT a l-value

l-value reference is a reference to l-value
r-value reference is a reference to r-value
-----------------------------------------------------------------*/
static void whatisRvalueReference() {
  {
    int a = 5; //a is l-value. 5 is a r-value
    int& b = a; //b is l-value reference to l-value
    //int&& c ; //c is a r-value reference to r-value


    p(a); //In static void p(int x)5
    p1(b); //In static void p1(int& x)5
    p1(6);//In static void p1(int&& x)6
  }
}

/*-----------------------------------------------------------------
                            ex1 starts
```

```
----------------------------------------------------------------*/

static int getx() {
  int x = 20;
  return x;
}

static void ex1() {
  int x = 10;
  p(x);
  p1(x);
  p(getx());
  //p1(getx()); //error C2664: 'void p1(int &)': cannot convert argument 1 from 'int' to 'int &'
  p1(56);
  p1(getx());
}

/*----------------------------------------------------------------
                              ex1 ends
----------------------------------------------------------------*/


/*----------------------------------------------------------------
                                whatIsTheUse starts
----------------------------------------------------------------*/
static void whatIsTheUse() {
  std::string str = "Hello";
  std::vector<std::string> v;
  v.push_back(str);
  std::cout << "str after pushing: " << str << std::endl;

  v.push_back(std::move(str));
  std::cout << "str after std::move(str) pushing: " << str << std::endl;

  {
    for (int i = 0; i < 100; ++i) {
      std::string str = "Hello";
      v.push_back(str);
    }
    for (int i = 0; i < 100; ++i) {
      std::string str = "Hello";
      v.push_back(std::move(str));
    }

  }
}

/*----------------------------------------------------------------
```

```
                                whatIsTheUse ends
-----------------------------------------------------------------*/
/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void testBed() {
  whatisLvalueAndRvalue();
  whatisRvalueReference();
  ex1();
  whatIsTheUse();
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testBed();
  return 0;
}
```

VASUDEVA

12.4 Int class

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

```
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: Int.h
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has Int class declaration
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#ifndef Int_H
#define Int_H

#include "../util/util.h"

//PLAY by commenting and un-commenting to understand l-vale and r-value
#define MOVECONSTRUCTOR
#define MOVEEQUALOPERATOR

/*----------------------------------------------------------------
Declaration of Int class
----------------------------------------------------------------*/
class Int {
public:
  Int(int x = 0);
  ~Int();
  Int(const Int& b);
#ifdef MOVECONSTRUCTOR
  Int(Int&& b); //Move constructor
#endif
  Int& operator=(const Int& b);
#ifdef MOVEEQUALOPERATOR
  Int& operator=(Int&& b);
#endif
  int getd() const { return d_; }
  void change(int d);

  friend ostream& operator<<(ostream& o, const Int& a);
  friend Int operator+(const Int& a, const Int& b);
  friend Int operator-(const Int& a, const Int& b);
  friend bool operator==(const Int& a, const Int& b) { return (a.d_ == b.d_); }
  friend bool operator<(const Int& a, const Int& b) { return (a.d_ < b.d_); }
  friend bool operator>(const Int& a, const Int& b) { return (a.d_ > b.d_); }

  static bool show() { return show_; }
  static void setDisplay(bool x) { show_ = x; }

```

```
50 private:
51   //Private data
52   int d_;
53   string* pd_;
54   static bool show_;
55   //Private functions
56   void copy_(const Int& b);
57   void allocate_(int x);
58   void release_();
59 };
60
61 /*-----------------------------------------------------------------
62 functions for containers
63 -----------------------------------------------------------------*/
64 struct ascendingInt {
65   bool operator() (const Int& a, const Int& b) {
66     return (a < b) ;
67   }
68 };
69
70 struct descendingInt {
71   bool operator() (const Int& a, const Int& b) {
72     return (a > b) ;
73   }
74 };
75
76 struct printInt {
77   void operator() (const Int& a) {
78     cout << a << " " ;
79   }
80 };
81
82 struct add3Int {
83   void operator() (Int& a) {
84     a = a + 3 ;
85   }
86 };
87
88 struct displaySignInt {
89   void operator() (const Int& a) {
90     if (a.getd() > 0) {
91       std::cout << "+" ;
92     }
93     std::cout << a << " " ;
94   }
95 };
96
97 struct ascendingIntPtr {
98   bool operator() (Int* const a, Int* const b) {
```

```
    return (*a < *b) ;
  }
};

struct descendingIntPtr {
  bool operator() (Int* const a, Int* const b) {
    return (*a > *b) ;
  }
};

struct printIntPtr {
  void operator() (Int* const a) {
    cout << *a << " " ;
  }
};

struct add3IntPtr {
  void operator() (Int* a) {
    *a = *a + 3 ;
  }
};

struct displaySignIntPtr {
  void operator() (Int* const a) {
    if (a->getd() > 0) {
      std::cout << "+" ;
    }
    std::cout << *a << " " ;
  }
};

struct deleteInt {
  void operator() (Int*  a) {
    delete a ;
  }
};

#endif
//EOF
```

```
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: Inttest.cpp

On linux:
g++ util.cpp Int.cpp Inttest.cpp
valgrind a.out
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test Int object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "Int.h"

/*----------------------------------------------------------------
To teach basic
----------------------------------------------------------------*/
static void basic() {
  Int a(10);
  std::cout << "a = " << a << endl;
  Int b(a);
  std::cout << "b = " << b << endl;
  Int c(20);
  std::cout << "c = " << c << endl;
  b = c;
  std::cout << "b = " << b << endl;
  std::cout << "c = " << c << endl;
}

/*----------------------------------------------------------------
pass_by_value
----------------------------------------------------------------*/
static void pass_by_value(Int a) {
  std::cout << "a = " << a << endl;
  a.change(a.getd() + 5);
  std::cout << "a = " << a << endl;
}

/*----------------------------------------------------------------
pass_by_reference
----------------------------------------------------------------*/
static void pass_by_reference(Int& a) {
  std::cout << "a = " << a << endl;
  a.change(a.getd() + 5);
  std::cout << "a = " << a << endl;
```

```
}

/*-----------------------------------------------------------------
pass_by_move
-----------------------------------------------------------------*/
static void pass_by_move(Int&& a) {
  std::cout << "a = " << a << endl;
  a.change(a.getd() + 5);
  std::cout << "a = " << a << endl;
}

/*-----------------------------------------------------------------
To teach pass by value
-----------------------------------------------------------------*/
static void how_to_pass() {
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    pass_by_value(a);
    std::cout << "a = " << a << endl;
  }
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    pass_by_reference(a);
    std::cout << "a = " << a << endl;
  }
  {
    pass_by_move(Int(20));
  }
}

/*-----------------------------------------------------------------
return_by_value
-----------------------------------------------------------------*/
static Int return_by_value() {
  Int a(10);
  std::cout << "a = " << a << endl;
  return a;
}

/*-----------------------------------------------------------------
To teach need of move constructor
-----------------------------------------------------------------*/
static void why_move_constructor_is_required() {
  {
    //RUN THIS ROUTINE BY COMMENTING MOVECONSTRUCTOR
    Int a = return_by_value();
    std::cout << "a = " << a << endl;
```

```
  }
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void equal_operator() {
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    Int b(20);
    std::cout << "b = " << b << endl;
    Int c(30);
    std::cout << "c = " << c << endl;
    a = b = c;
    std::cout << "a = " << a << endl;
    std::cout << "b = " << b << endl;
    std::cout << "c = " << c << endl;
  }
}

/*-----------------------------------------------------------------
swap
-----------------------------------------------------------------*/
static void swap(Int& a, Int& b) {
  Int t = a;
  a = b;
  b = t;
}

/*-----------------------------------------------------------------
swap
-----------------------------------------------------------------*/
static void swapMove(Int& a, Int& b) {
  Int t = std::move(a);
  a = std::move(b);
  b = std::move(t);
}


/*-----------------------------------------------------------------
RUN THIS ROUTINE BY COMMENTING BOTH
#define MOVECONSTRUCTOR
#define MOVEEQUALOPERATOR
-----------------------------------------------------------------*/
static void why_move_equal_operator_is_required() {
  {
    Int a(10);
    std::cout << "a = " << a << endl;
```

```
    Int b(20);
    std::cout << "b = " << b << endl;
    swap(a, b);
    std::cout << "a = " << a << endl;
    std::cout << "b = " << b << endl;
  }
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    Int b(20);
    std::cout << "b = " << b << endl;
    swapMove(a, b);
    std::cout << "a = " << a << endl;
    std::cout << "b = " << b << endl;
  }
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void testBed() {
  Int::setDisplay(true);
  basic();
  how_to_pass();
  why_move_constructor_is_required(); //To teach need of move constructor
  equal_operator();
  why_move_equal_operator_is_required();
}
/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testBed();
  return 0;
}

//EOF
```

```
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: Inttest.cpp

On linux:
g++ util.cpp Int.cpp Inttest.cpp
valgrind a.out
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test Int object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "Int.h"

/*----------------------------------------------------------------
To teach basic
----------------------------------------------------------------*/
static void basic() {
  Int a(10);
  std::cout << "a = " << a << endl;
  Int b(a);
  std::cout << "b = " << b << endl;
  Int c(20);
  std::cout << "c = " << c << endl;
  b = c;
  std::cout << "b = " << b << endl;
  std::cout << "c = " << c << endl;
}

/*----------------------------------------------------------------
pass_by_value
----------------------------------------------------------------*/
static void pass_by_value(Int a) {
  std::cout << "a = " << a << endl;
  a.change(a.getd() + 5);
  std::cout << "a = " << a << endl;
}

/*----------------------------------------------------------------
pass_by_reference
----------------------------------------------------------------*/
static void pass_by_reference(Int& a) {
  std::cout << "a = " << a << endl;
  a.change(a.getd() + 5);
  std::cout << "a = " << a << endl;
```

```
}

/*-----------------------------------------------------------------
pass_by_move
-----------------------------------------------------------------*/
static void pass_by_move(Int&& a) {
  std::cout << "a = " << a << endl;
  a.change(a.getd() + 5);
  std::cout << "a = " << a << endl;
}

/*-----------------------------------------------------------------
To teach pass by value
-----------------------------------------------------------------*/
static void how_to_pass() {
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    pass_by_value(a);
    std::cout << "a = " << a << endl;
  }
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    pass_by_reference(a);
    std::cout << "a = " << a << endl;
  }
  {
    pass_by_move(Int(20));
  }
}

/*-----------------------------------------------------------------
return_by_value
-----------------------------------------------------------------*/
static Int return_by_value() {
  Int a(10);
  std::cout << "a = " << a << endl;
  return a;
}

/*-----------------------------------------------------------------
To teach need of move constructor
-----------------------------------------------------------------*/
static void why_move_constructor_is_required() {
  {
    //RUN THIS ROUTINE BY COMMENTING MOVECONSTRUCTOR
    Int a = return_by_value();
    std::cout << "a = " << a << endl;
```

```
  }
}

/*----------------------------------------------------------------

----------------------------------------------------------------*/
static void equal_operator() {
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    Int b(20);
    std::cout << "b = " << b << endl;
    Int c(30);
    std::cout << "c = " << c << endl;
    a = b = c;
    std::cout << "a = " << a << endl;
    std::cout << "b = " << b << endl;
    std::cout << "c = " << c << endl;
  }
}

/*----------------------------------------------------------------
swap
----------------------------------------------------------------*/
static void swap(Int& a, Int& b) {
  Int t = a;
  a = b;
  b = t;
}

/*----------------------------------------------------------------
swap
----------------------------------------------------------------*/
static void swapMove(Int& a, Int& b) {
  Int t = std::move(a);
  a = std::move(b);
  b = std::move(t);
}


/*----------------------------------------------------------------
RUN THIS ROUTINE BY COMMENTING BOTH
#define MOVECONSTRUCTOR
#define MOVEEQUALOPERATOR
----------------------------------------------------------------*/
static void why_move_equal_operator_is_required() {
  {
    Int a(10);
    std::cout << "a = " << a << endl;
```

```
    Int b(20);
    std::cout << "b = " << b << endl;
    swap(a, b);
    std::cout << "a = " << a << endl;
    std::cout << "b = " << b << endl;
  }
  {
    Int a(10);
    std::cout << "a = " << a << endl;
    Int b(20);
    std::cout << "b = " << b << endl;
    swapMove(a, b);
    std::cout << "a = " << a << endl;
    std::cout << "b = " << b << endl;
  }
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
static void testBed() {
  Int::setDisplay(true);
  basic();
  how_to_pass();
  why_move_constructor_is_required(); //To teach need of move constructor
  equal_operator();
  why_move_equal_operator_is_required();
}
/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testBed();
  return 0;
}

//EOF
```

12.5 vector class

Constructors

vector<T> a	Default constructor	O(1)
vector<T> a(size)	Default constructor with explicit size	O(size)
vector<T> a(size,T)	Default constructor with explicit size and initial value	O(size)

Size

a.size()	Number of elements currently held in vector a	O(1)
a.capacity()	Maximum elements vector a can hold	O(1)
a.clear()	All elements are removed size = 0 and capacity = unchanged	O(n)

Element access Returns object by alias

a[i]	Access i th element of array a i must be less than size of a boundary checking is not done	O(1)
a.at[i]	Access i th element of array a Boundary check is done before access. Throws exception if (i < 0) || (i >= size)	O(1)
a.push_back(i)	Add element to the end of the array a.	O(1) OR O(n)
a.pop_back()	Remove element from the end of the array a.	O(1)

Figure 12.1: **vector** class

VASUDEVAMURTHY

```
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: svector.h
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has svector class declaration
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#ifndef svector_H
#define svector_H

#include "../../Int/Int.h"

/*----------------------------------------------------------------
Declaration of svector class
----------------------------------------------------------------*/
class svector {
public:
  void why_vector_is_required();
  void basic();
  void understanding_for_each_int();
  void basicUDT();
  void understanding_for_each_UDT();
  void basicUDTPointer();
  void array_as_vector();
  void can_you_find_the_bug();
private:
  template <class T> void p_(const string& t, const vector<T>& a);
  template <class T> void understanding_iterator_(vector<T>& a);
  template <class T> void understanding_lower_bound_and_upper_bound_(vector<T>& a);
  void understand_passing_UDT_();
  void difference_between_push_back_and_emplace_back_();
  void pass_(vector<Int> d);
  void pass1_(vector<Int>& a);
  vector<Int>  pass1_(vector<Int>&& a);
};

#endif
//EOF
```

Figure 12.2: Concept of **iterators**

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: svector.cpp

On linux
g++ ../../Int/Int.cpp  svector.cpp svectortest.cpp
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file has svector class definition
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "svector.h"


/*----------------------------------------------------------------
Definition of routines of svector class
----------------------------------------------------------------*/

/*----------------------------------------------------------------
Why Vector is required?
1. Array is a homogenous container
2. Vector is a hetrogenous container
----------------------------------------------------------------*/
void svector::why_vector_is_required() {
  {
    //Static allocation
    int a[10] = { 99 };
    //Problem 1: 10 must be known at the time of compilation
    //Problem 2: Can store only int
    //Problem 3: When you pass array a, you need to pass size also. Callee will NOT know size
  }
  {
    //Dynamic allocation
    int n = 9897;
    int* a = new int(n);
    //Problem 1: n is dynamic. But n cannot be changed after allocation
    //Problem 2: Can store only int
    //Problem 3: When you pass array a, you need to pass size also. Callee will NOT know size
    delete a;
  }

  {
    //Container
```

```
    vector <int> a;
    vector <Int> b;
    vector <Int*> c;
    //Advantage 1: Need to have to commit to a size
    //Advantage 2: Can store ANY object including basic type. Java cannot do this
    //Advantage 1: When you pass array a, you need NOT have to pass size also.
  }
}

/*-----------------------------------------------------------------
Generic p_
Why this routine is Templated?
-----------------------------------------------------------------*/
template <class T>
void svector::p_(const string& t, const vector<T>& a) {
  cout << t << endl;
  cout << "-----------------" << endl;
  cout << "size = " << a.size() << " ";
  cout << "capacity = " << a.capacity() << " ";
  cout << endl;
  {
    //You think continuous piece of data
    auto size = a.size();
    for (int i = 0; i < size; ++i) {
      cout << "a[" << i << "]= " << a[i] << " ";
    }
    cout << endl;
  }
  {
    //You don't care how data is stored
    //You say give each data e in the container
    int i = 0;
    for (const T& e : a) {  //Note reference
      cout << "a[" << i++ << "]= " << e << " ";
    }
    cout << endl;
  }
  cout << "-----------------" << endl;
}

/*-----------------------------------------------------------------
array_as_vector
-----------------------------------------------------------------*/
void svector::array_as_vector() {
  vector<int> a(3, -99);
  //You can now access at arbitrary location without push_back
  p_("array of size 3 before ", a);
  a[1] = 100;
```

```
  a[0] = -100;
  a[2] = 200;
  p_("array of size 3 after", a);
}

/*-----------------------------------------------------------------
a[0] ..... a[9]

begin() will point to a[0]
end() will NOT POINT to a[9], but to one element past a[9]
That means real end is: end()-1 ;

Why this routine is Templated?
-----------------------------------------------------------------*/
template <typename T>
void svector::understanding_iterator_(vector<T>& a) {
  {
    cout << "Understanding forward traversal " << endl;
    auto itt = a.begin();
    while (itt != a.end()) {
      cout << *itt << " ";
      ++itt;
    }
    cout << endl;
  }

  {
    cout << "Understanding reverse traversal " << endl;
    auto itt = a.end();
    while (itt != a.begin()) {
      --itt;
      cout << *itt << " ";
    }
    cout << endl;
  }

  {
    cout << "Understanding frward traversal with changing contents" << endl;
    auto itt = a.begin();
    while (itt != a.end()) {
      *itt = *itt + 5;
      ++itt;
    }
    p_("after adding each number by 5", a);
  }

  {
    cout << "Understanding reverse traversal with changing contents" << endl;
    auto itt = a.end();
```

```
    while (itt != a.begin()) {
      --itt;
      *itt = *itt - 5;
    }
    p_("after subtracting each number by 5", a);
  }
}

/*-----------------------------------------------------------------
lower_bound returns an iterator pointing to the first element in the
 range [first,last) which has a value not less than ‘val’.

upper_bound returns an iterator pointing to the first element in the
 range [first,last) which has a value greater than ‘val’.

 0 1 2 3 4 5 6 7
 5,6,7,7,6,5,5,6
 lower_bound for 6 is at position 1 (at position 1 we have a number 6 not less than 6
 upper_bound for 6 is at position 8

 0 1 2 3 4 5 6 7
 5 5 5 6 6 6 7 7

 lower_bound for 6 is at position 3
 upper_bound for 6 is at position 6

This can be done in O(log n) Intity if sorted
If it is NOT sorted it requires O(n)

Why this routine is Templated?
-----------------------------------------------------------------*/
template <typename T>
void svector::understanding_lower_bound_and_upper_bound_( vector<T>& a) {
  p_("Given data", a);
  int k = 6;
  {
    auto lower = lower_bound(a.begin(), a.end(), k);
    cout << "lower_bound for " << k << " is at position " << (lower - a.begin()) << endl;
  }
  {
    auto upper = upper_bound(a.begin(), a.end(), k);
    cout << "upper_bound for " << k << " is at position " << (upper - a.begin()) << endl;
  }

  sort(a.begin(), a.end());
  p_("After sort", a);
```

```
  {
    auto lower = lower_bound(a.begin(), a.end(), k);
    cout << "lower_bound for " << k << " is at position " << (lower - a.begin
     ()) << endl;
  }
  {
    auto upper = upper_bound(a.begin(), a.end(), k);
    cout << "upper_bound for " << k << " is at position " << (upper - a.begin
     ()) << endl;
  }
}

/*-----------------------------------------------------------------
basic routines on int
-----------------------------------------------------------------*/
void svector::basic() {
  vector<int> a;
  p_("Vector a start", a); //DO NOT commit any capacity

  vector<int> a1(3); //Start with a capacity of 3
  p_("Vector a1 start", a1);

  vector<int> c(3, -99); //Start with a capacity of 3. All initalized to 99
  p_("Vector c start", c);

  vector<int> b{ 10,20,40,67,99 };
  p_("Vector b start", b);

  understanding_iterator_(b);
  vector<int> d{ 5,6,7,7,6,5,5,6 };
  understanding_lower_bound_and_upper_bound_(d);

  p_("before pushing 7 numbers", b);
  for (int i = 0; i < 7; ++i) {
    b.push_back(i); //YOU CAN ADD ONLY AT THE END
  }
  p_("after pushing 7 numbers", b);

  sort(b.begin(), b.end(), ascendingInt()); ;
  p_("after ascending sorting b", b);

  auto b20 = lower_bound(b.begin(), b.end(), 20);
  if (b20 != b.end()) {
    b.erase(b20);
    p_("after removing 20 from b", b);
  }

  auto b200 = lower_bound(b.begin(), b.end(), 200);
  if (b200 != b.end()) {
```

```
    b.erase(b200);
    p_("after removing 20 from b", b);
  } else {
    cout << "200 is not in array\n";
  }

  sort(b.begin(), b.end(), descendingInt());
  p_("after descending sorting b", b);
}

/*-----------------------------------------------------------------
Understanding pass by value, l-value reference and r-value-reference
-----------------------------------------------------------------*/
void svector::pass_(vector<Int> a) {
  p_("In pass_(vector<Int> a)", a);
  for_each(a.begin(), a.end(), add3Int());
  p_("In pass_(vector<Int> a) after adding 3", a);
}

void svector::pass1_(vector<Int>& a) {
  p_("In pass1_(vector<Int>& a)", a);
  for_each(a.begin(), a.end(), add3Int());
  p_("pass1_(vector<Int>& a) after adding 3", a);
}

vector<Int> svector::pass1_(vector<Int>&& a) {
  p_("pass1_(vector<Int>&& a)", a);
  for_each(a.begin(), a.end(), add3Int());
  p_("pass1_(vector<Int>&& a) after adding 3", a);
  return a;
}

/*-----------------------------------------------------------------
When we use push_back(), we create an object and then insert it into the vector.
With emplace_back(), the object is constructed in-place and saves an unnecessary copy.

Teach this with commenting
//#define MOVECONSTRUCTOR
//#define MOVEEQUALOPERATOR
-----------------------------------------------------------------*/
void svector::difference_between_push_back_and_emplace_back_() {
  {
    vector<Int> a;
    //create enough capacity so that it will not grow
    for (int i = 0; i < 100; ++i) {
      a.push_back(0);
    }
```

```
    for (int i = 0; i < 5; ++i) {
      cout << "size = " << a.size() << " ";
      cout << "capacity = " << a.capacity() << " ";
      cout << endl;
      a.push_back(i);
    }
  }
  {
    vector<Int> a;
    //create enough capacity so that it will not grow
    for (int i = 0; i < 100; ++i) {
      a.push_back(0);
    }
    for (int i = 0; i < 5; ++i) {
      cout << "size = " << a.size() << " ";
      cout << "capacity = " << a.capacity() << " ";
      cout << endl;
      a.emplace_back(i);
    }
  }
}

void svector::understand_passing_UDT_() {
  {
    vector<Int> a(3, -99);
    pass_(a);
    p_("a after pass_(a)", a);
  }
  {
    vector<Int> a(3, -99);
    pass1_(a);
    p_("a after pass1_(a)", a);
  }
  {
    //Behaviour is safe whether I comment or not
    vector<Int> a = pass1_(vector<Int>(3, -99));
    p_("after vector<Int> a pass1_", a);
  }
}

/*-----------------------------------------------------------------
basic routines on UDT
-----------------------------------------------------------------*/
void svector::basicUDT() {
  vector<Int> a;
  p_("Vector a start", a); //DO NOT commit any capacity

  vector<Int> a1(3); //Start with a capacity of 3
```

```
  p_("Vector a1 start", a1);

  vector<Int> c(3, -99); //Start with a capacity of 3. All initalized to 99
  p_("Vector c start", c);


  vector<Int> b{ 10,20,40,67,99 };
  p_("Vector b start", b);


  understanding_iterator_(b);
  vector<int> d{ 5,6,7,7,6,5,5,6 };
  understanding_lower_bound_and_upper_bound_(d);


  for (int i = 0; i < 7; ++i) {
    b.push_back(i);
  }
  p_("after pushing 7 numbers", b);


  sort(b.begin(), b.end(), ascendingInt()); ;
  p_("after ascending sorting b", b);


  auto b20 = lower_bound(b.begin(), b.end(), 20);
  if (b20 != b.end()) {
    b.erase(b20);
    p_("after removing 20 from b", b);
  }

  auto b200 = lower_bound(b.begin(), b.end(), 200);
  if (b200 != b.end()) {
    b.erase(b200);
    p_("after removing 20 from b", b);
  } else {
    cout << "200 is not in array\n";
  }

  sort(b.begin(), b.end(), descendingInt());
  p_("after descending sorting b", b);
}

/*-------------------------------------------------------------

first, last - the range to apply the function to

f - function object, to be applied to the result of
dereferencing every iterator in the range [first, last)
```

```

----------------------------------------------------------------*/
void svector::understanding_for_each_int() {
  {
    vector<int> a{ 10,20,40,67,99 };
    for_each(a.begin(), a.end(), printint());
    cout << endl;
    for_each(a.begin(), a.end(), add3int());
    cout << "After adding 3 to the entire array\n";
    for_each(a.begin(), a.end(), printInt());
    cout << endl;
  }
  {
    vector<int> a{ -3,-2,-1,0,1,2,3,4,5,6 };
    for_each(a.begin(), a.end(), displaySignInt());
    cout << endl;
    auto s = a.begin();
    auto e = a.end();
    auto r1 = next(s, 2); //r1 is at -1
    auto r2 = prev(e, 3); //r2  is at 3
    for_each(r1, r2, displaySignInt());
    cout << endl;
  }
}



/*----------------------------------------------------------------

first, last - the range to apply the function to

f - function object, to be applied to the result of
dereferencing every iterator in the range [first, last)

----------------------------------------------------------------*/
void svector::understanding_for_each_UDT() {
  {
    vector<Int> a{ 10,20,40,67,99 };
    for_each(a.begin(), a.end(), printInt());
    cout << endl;
    for_each(a.begin(), a.end(), add3Int());
    cout << "After adding 3 to the entire array\n";
    for_each(a.begin(), a.end(), printInt());
    cout << endl;
  }
  {
    vector<Int> a{ -3,-2,-1,0,1,2,3,4,5,6 };
    for_each(a.begin(), a.end(), displaySignInt());
    cout << endl;
```

```
    auto s = a.begin();
    auto e = a.end();
    auto r1 = next(s, 2); //r1 is at -1
    auto r2 = prev(e, 3); //r2  is at 3
    for_each(r1, r2, displaySignInt());
    cout << endl;
  }
  {
    understand_passing_UDT_();
  }
  {
    difference_between_push_back_and_emplace_back_();
  }
}

/*-----------------------------------------------------------------
basic routines on UDT pointer
-----------------------------------------------------------------*/
void svector::basicUDTPointer() {
  vector<Int*> a;
  p_("Vector a start", a); //DO NOT commit any capacity

  vector<Int*> c(3); //start with 3
  p_("Vector c start", c);
  for (int i = 0; i < 10; ++i) {
    Int* p = new Int(i);
    c.push_back(p);
  }
  p_("Vector c after adding 10 Int", c);


  vector<Int*> b;
  for (int i : {10, 20, 40, 67, 99}) {
    b.push_back(new Int(i));
  }
  p_("Vector b start", b);

  understanding_iterator_(b);

  for (int i = 0; i < 7; ++i) {
    b.push_back(new Int(i));
  }
  p_("after pushing 7 numbers", b);


  sort(b.begin(), b.end(), ascendingIntPtr()); ;
  p_("after ascending sorting b", b);
  for_each(b.begin(), b.end(), printIntPtr());
  cout << endl;
```

```
  //DO NOT USE lower_bound on pointer array
  if (0) {
    Int* p200ptr = new Int(200);
    {
      auto b200 = lower_bound(b.begin(), b.end(), p200ptr);
      //MUST REBUILD if u change anything
      if (b200 != b.end()) {
        b.erase(b200);
        p_("after removing 200 from b", b);
      } else {
        cout << "b200ptr is not in array\n";
      }
    }
    b.push_back(p200ptr);
    for_each(b.begin(), b.end(), printIntPtr());
    cout << endl;
    {
      //DO NOT USE lower_bound on pointers array
      auto b200 = lower_bound(b.begin(), b.end(), p200ptr);
      if (b200 != b.end()) {
        b.erase(b200);
        p_("after removing 200  from b", b);
        for_each(b.begin(), b.end(), printIntPtr());
        cout << endl;
      } else {
        cout << "200 is not in array\n";
      }
    }
  }
  sort(b.begin(), b.end(), descendingIntPtr());
  p_("after descending sorting b", b);
  for_each(b.begin(), b.end(), printIntPtr());
  cout << endl;

  if (1) { //make 0 to see memory leaks
    //Now release all memory
    for_each(a.begin(), a.end(), deleteInt());
    for_each(b.begin(), b.end(), deleteInt());
    for_each(c.begin(), c.end(), deleteInt());
  }
}

/*-----------------------------------------------------------------
problem

MORAL: DO NOT STORE ALIAS TO THE CONTAINER OBJECT
-----------------------------------------------------------------*/
void svector::can_you_find_the_bug() {
  vector<Int> vc;
```

```
  p_("begin with ", vc);
  vc.push_back(0);
  vc.push_back(1);
  vc.push_back(2);

  p_("After Inserting 0, 1 and 2 ", vc);

  Int& c1 = vc[1];
  cout << " c1 = " << c1 << endl;

  for (int i = 3; i < 10; ++i) {
    vc.push_back(i);
  }
  p_("After Inserting  3 to 10 ", vc);

  //WHAT HAPPENS HERE ????????????
  if (0) {
    cout << " c1 = " << c1 << endl;
  }
}
//EOF
```

VASUDEV. HV

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
file: svectortest.cpp

On linux:
g++ ../../Int/Int.cpp  svector.cpp svectortest.cpp
valgrind a.out
----------------------------------------------------------------*/

/*----------------------------------------------------------------
This file test svector object
----------------------------------------------------------------*/

/*----------------------------------------------------------------
All includes here
----------------------------------------------------------------*/
#include "svector.h"

/*----------------------------------------------------------------
RUN this first by commenting
//#define MOVECONSTRUCTOR
//#define MOVEEQUALOPERATOR

Run again with
#define MOVECONSTRUCTOR
#define MOVEEQUALOPERATOR

rebuild solution
----------------------------------------------------------------*/
static void testbed() {
  {
    svector a;
    a.why_vector_is_required();
  }
  {
    svector a;
    a.basic();
    a.understanding_for_each_int();
  }
  {
    svector a;
    a.array_as_vector();
  }
  {
    svector a;
    a.basicUDT();
    a.understanding_for_each_UDT();
  }
  {
```

```
    svector a;
    a.basicUDTPointer();
  }
  {
    svector a;
    a.can_you_find_the_bug();
  }
}

/*-----------------------------------------------------------------

-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  Int::setDisplay(true);
  testbed();
  return 0;
}

//EOF


```

VASUDEV. HV

12.6 string class

VASUDEVAMURTHY

```cpp
/*----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: string.cpp
compile: g++ sting.cpp
----------------------------------------------------------------*/
#include <iostream>
#include <string>
#include <vector>

using namespace std;
#ifdef _MSC_VER
#pragma warning(disable: 4996) /* Disable deprecation */
#endif

/*----------------------------------------------------------------
print1
----------------------------------------------------------------*/
static void print1(const char* title, const string& s) {
  cout << title;
  cout << s << endl;
  cout << "size = " << s.size() << endl;
}

/*----------------------------------------------------------------
basic tests
s4 is an empty string
s = : abcd
size = 4
s[3] = d
s1 = : abcd
size = 4
s1 == s
s != abc
s = : abcdbabu
size = 8
s = : abcdbabuz
size = 9
s = : abcdbabuzT
size = 10
s = : abcdbabuzT78
size = 12
s after clear:
size = 0
----------------------------------------------------------------*/
static void test_basic() {
  string s4;
  /* How to test the string is empty */
  if (s4.empty()) {
    cout << "s4 is an empty string" << endl;
```

```
  }
  string s = "abcd";
  print1("s = : ", s);
  cout << "s[3] = " << s[3] << endl;
  string s1(s);
  print1("s1 = : ", s1);
  if (s1 == s) {
    cout << "s1 == s" << endl;
  }
  if (s != "abc") {
    cout << "s != abc " << endl;
  }
  s = s + "babu";
  print1("s = : ", s);
  s = s + 'z';
  print1("s = : ", s);
  s.append("T");
  print1("s = : ", s);
  /* How to append an integer to string */
  int x = 78;
  s = s + to_string(x) ;
  print1("s = : ", s);
  /* How to clear a string */
  s.clear();
  print1("s after clear: ", s);
}

/*-----------------------------------------------------------------
testing cin

Enter your name: jag murthy phd
s after getline >> s: jag murthy phd
size = 14
Enter your Country: san jose ca
s after getline >> s: san jose ca
size = 11
-----------------------------------------------------------------*/
static void test_cin() {
  string s;
  cout << "Enter your name: ";
  getline(cin, s);
  print1("s after getline >> s: ", s);
  cout << "Enter your Country: ";
  getline(cin, s);
  print1("s after getline >> s: ", s);
}

/*-----------------------------------------------------------------
testing iterators
```

```

a = : A quick brown fix jumps over a lazy dog
size = 39
Understanding forward traversal
A   q u i c k   b r o w n   f i x   j u m p s   o v e r   a   l a z y   d o g
Understanding reverse traversal
g o d   y z a l   a   r e v o   s p m u j   x i f   n w o r b   k c i u q   A
-----------------------------------------------------------------*/
static void test_iterators() {
  string a("A quick brown fix jumps over a lazy dog");
  print1("a = : ", a);
  {
    cout << "Understanding forward traversal" << endl;
    string::iterator itt = a.begin();
    while (itt != a.end()) {
      cout << *itt << " ";
      itt++;
    }
    cout << endl;
  }
  {
    cout << "Understanding forward traversal using auto" << endl;
    auto itt = a.begin();
    while (itt != a.end()) {
      cout << *itt << " ";
      itt++;
    }
    cout << endl;
  }
  {
    cout << "Understanding forward traversal using range" << endl;
    for (const char& c : a) {
      cout << c << " ";
    }
    cout << endl;
  }
  {
    cout << "Understanding reverse traversal" << endl;
    string::iterator itt = a.end();
    //cout << *itt << endl ; ----------- CORE DUMPS
    while (itt != a.begin()) {
      itt--; //decrement iterator before using
      cout << *itt << " ";
    }
    cout << endl;
  }
  {
    cout << "Understanding reverse traversal using auto" << endl;
    auto itt = a.end();
```

```
    //cout << *itt << endl ; ----------- CORE DUMPS
    while (itt != a.begin()) {
      itt--; //decrement iterator before using
      cout << *itt << " ";
    }
    cout << endl;
  }
}

/*-----------------------------------------------------------------
testing find

str: Hello, can you find UCSC ext? at Santa clara UCSC
size = 49
First occurrence of UCSC was found at: 20
-----------------------------------------------------------------*/
static void test_find() {
  string str("Hello, can you find UCSC ext? at Santa clara UCSC");
  string::size_type position = str.find("UCSC");
  print1("str: ", str);
  cout << "First occurrence of UCSC was found at: " << position << endl;
}

/*-----------------------------------------------------------------
"A quick brown fix jumps over a lazy dog"
token[0] = A
token[1] = quick
token[2] = brown
token[3] = fix
token[4] = jumps
token[5] = over
token[6] = a
token[7] = lazy
token[8] = dog
-----------------------------------------------------------------*/
static int tokenize(const string& str, vector<string>& tokens, const string& delimiters = " ") {
  // Skip delimiters at beginning.
  string::size_type lastPos = str.find_first_not_of(delimiters, 0);
  // Find first "non-delimiter".
  string::size_type pos = str.find_first_of(delimiters, lastPos);

  while (string::npos != pos || string::npos != lastPos) {
    // Found a token, add it to the vector.
    tokens.push_back(str.substr(lastPos, pos - lastPos));
    // Skip delimiters.  Note the "not_of"
    lastPos = str.find_first_not_of(delimiters, pos);
    // Find next "non-delimiter"
    pos = str.find_first_of(delimiters, lastPos);
```

```
  }
  return int(tokens.size());
}

/*-----------------------------------------------------------------
"UCSC extension|UCSC|California|United States of America|North America|World", "|"

token[0] = UCSC extension
token[1] = UCSC
token[2] = California
token[3] = United States of America
token[4] = North America
token[5] = World
-----------------------------------------------------------------*/
static void test_tokenizer(const string& str, const string& delimiters = " ") {
  vector<string> tokens;
  int size = tokenize(str, tokens, delimiters);
  auto itt = tokens.begin();
  int i = 0;
  while (itt != tokens.end()) {
    string& s = *itt;
    cout << "token[" << i++ << "] = " << s << endl;
    itt++;
  }
}


/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  test_basic();
  test_cin();
  test_iterators();
  test_find();
  test_tokenizer("A quick brown fix jumps over a lazy dog");
  test_tokenizer("UCSC extension|UCSC|California|United States of America|North America|World", "|");
  return EXIT_SUCCESS;
}

```

12.7 deque class

VASUDEVAMURTHY

Figure 12.3: **deque** class

VASUDEVAMURTHY

```
{
  cout << "Understanding forward traversal" << endl;
  auto itt = a.begin();
  while (itt != a.end()) {
    cout << *itt << " ";
    ++itt;
  }
  cout << endl;
}
{
  cout << "Understanding reverse traversal " << endl;
  auto itt = a.end();
  //cout << *itt << endl ; ------------ CORE DUMPS
  while (itt != a.begin()) {
    --itt; //decrement iterator before using
    cout << *itt << " ";
  }
  cout << endl;
}
```

Figure 12.4: **deque** class

VASUDEVAMURTHY

```
/*-----------------------------------------------------------------
Copyright (c) 2019 Author: Jagadeesh Vasudevamurthy
Filename: sdeque.cpp
compile: g++  ../../Int/Int.cpp sdeque.cpp
Memory leaked: 0 bytes (0%); potentially leaked: 0 bytes (0%)
-----------------------------------------------------------------*/
#include <iostream>
#include <deque>
#include <stdexcept> //Without this catch will NOT work on Linux


#ifdef _WIN32
#include "..\..\Int\Int.h"
#else
#include "../../Int/Int.h"
#endif

/*-----------------------------------------------------------------
print1
-----------------------------------------------------------------*/
template <typename T>
static void print1(const char* s, const deque<T>& a) {
  cout << s << endl;
  cout << "-----------------" << endl;
  cout << "size = " << a.size() << " ";
  cout << endl;
  for (int i = 0; i < int(a.size()); i++) {
    cout << "a[" << i << "] = " << a[i] << " ";
  }
  cout << endl;
  cout << "-----------------" << endl;
}

/*-----------------------------------------------------------------
a[0] ..... a[9]

begin() will point to a[0]
end() will NOT POINT to a[9], but to one element past a[9]
That means real end is: end()-1 ;
-----------------------------------------------------------------*/
static void understanding_iterator(deque<int>& a) {
  {
    cout << "Understanding forward traversal" << endl;
    auto itt = a.begin();
    while (itt != a.end()) {
      cout << *itt << " ";
      itt++;
    }
    cout << endl;
```

VASUDEVAMURTHY

```
  }
  {
    cout << "Understanding forward traversal using range" << endl;
    for (const auto& i : a) {
      cout << i << " ";
    }
    cout << endl;
  }

  {
    cout << "Understanding reverse traversal " << endl;
    auto itt = a.end();
    //cout << *itt << endl ; ----------- CORE DUMPS
    while (itt != a.begin()) {
      itt--; //decrement iterator before using
      cout << *itt << " ";
    }
    cout << endl;
  }
  cout << "-----------------" << endl;
}

/*-----------------------------------------------------------------
Understanding access using
1. a[i]
2. a.at(i)  -- Same as a[i], but does out of bound checks
and throws exception that you can catch

-----------------------------------------------------------------*/
static void understanding_at_access(const deque<int>& a, int i) {
  //int p = a[i] ; //core dumps
  try {
    int p = a.at(i); //Throws exception. out of bound checks

  } catch (std::out_of_range) {
    cout << " accessing a[" << i << "] but size of array is " << a.size() << endl;
  }
  cout << "-----------------" << endl;
}

/*-----------------------------------------------------------------
a.front() -- Returns first element of the deque
a.back()  -- Returns last  element of the deque
a.push_front(v) -- v is inserted as the first element of the deque
a.push_back(v) -- v is inserted as the back element of the deque
a.pop_front() - First element of the deque is removed. Nothing is returned
a.pop_back() - Last element of the deque is removed. Nothing is returned
-----------------------------------------------------------------*/
```

```
static void understanding_access(deque<int>& a) {
  print1("begin with", a);
  for (int i = 0; i < 5; i++) {
    a.push_back(i);
  }
  print1("After Inserting 5 elements from the back", a);
  for (int i = 0; i < 5; i++) {
    a.push_front(i * 10 + 1);
  }
  print1("After Inserting 5 elements from the front", a);

  int& y = a.front();
  cout << "The front of the deque has " << y << endl;
  int& z = a.back();
  cout << "The back of the deque has " << z << endl;

  print1("After front and back operation", a);

  //y = a.pop_front() ; WRONG. pop cannot return ;
  a.pop_front();
  a.pop_front();
  print1("After Removing two elements from the front", a);
  a.pop_back();
  a.pop_back();
  print1("After Removing two elements from the back", a);

  int x = int(a.size()) - 1;
  y = a[x];
  cout << "Random access of a[" << x << "]= " << y << endl;
  understanding_at_access(a, int(a.size()));
}

/*-----------------------------------------------------------------
basic
-----------------------------------------------------------------*/
static void basic() {
  deque<int> a;
  print1("begin with", a);
  understanding_access(a);
  understanding_iterator(a);

  for_each(a.begin(), a.end(), add3int());
  cout << "After adding 3\n";
  for_each(a.begin(), a.end(), printint());
  cout << endl;

  a.clear();
  print1("Using clear", a);
}
```

```

/*-----------------------------------------------------------------
array of user defined type
-----------------------------------------------------------------*/
static void test_deque_of_udt() {
  deque<Int> a;
  print1("begin with", a);
  for (int i = 0; i < 5; i++) {
    a.push_back(Int(i));
  }
  print1("After Inserting 5 elements from the back", a);
  for (int i = 0; i < 5; i++) {
    a.push_front(Int((i * 10 + 1)));
  }
  print1("After Inserting 5 elements from the front", a);

  Int& y = a.front();
  cout << "The front of the deque has " << y << endl;
  Int& z = a.back();
  cout << "The back of the deque has " << z << endl;
  {
    /* see what happens if U get Int by value */
    Int vf = a.front();
  }

  //y = a.pop_front() ; WRONG. pop cannot return ;
  a.pop_front();
  a.pop_front();
  print1("After Removing two elements from the front", a);
  a.pop_back();
  a.pop_back();
  print1("After Removing two elements from the back", a);

  int x = int(a.size()) - 1;
  Int &zz = a[x];
  cout << "Random access of a[" << x << "]= " << zz << endl;

  deque<Int> b(a);
  //apply("Contents of deque b",b,print_Int) ;
  bool equal = true;
  for (int i = 0; i < int(a.size()); i++) {
    if (a[i] != b[i]) {
      equal = false;
      break;
    }
  }
  equal ? cout << " array a == b\n " : cout << " array a != b\n ";
}

```

```
/*-----------------------------------------------------------------
array of user defined pointer type
-----------------------------------------------------------------*/
static void test_deque_of_ptr_udt() {
  deque<Int*> a;
  print1("begin with", a);
  for (int i = 0; i < 5; i++) {
    a.push_back(new Int(i));
  }
  cout << "After Inserting 5 elements from the back\n";
  for_each(a.begin(), a.end(), printIntPtr());
  for (int i = 0; i < 5; i++) {
    a.push_front(new Int(i * 10 + 1));
  }
  cout << "After Inserting 5 elements from the front\n";
  for_each(a.begin(), a.end(), printIntPtr());
  Int*& af = a.front();
  cout << "The front of the deque has " << *af << endl;
  Int*& ab = a.back();
  cout << "The back of the deque has " << *ab << endl;

  //y = a.pop_front() ; WRONG. pop cannot return ;
  delete(a.front());
  a.pop_front();

  delete(a.front());
  a.pop_front();

  cout << "After Removing two elements from the front\n";
  for_each(a.begin(), a.end(), printIntPtr());

  delete(a.back());
  a.pop_back();

  delete(a.back());
  a.pop_back();

  cout << "After Removing two elements from the back\n";
  for_each(a.begin(), a.end(), printIntPtr());
  deque<Int*> b(a);
  cout << "Contents of deque b:\n";
  for_each(a.begin(), a.end(), printIntPtr());
  for (int i = 0; i < int(a.size()); i++) {
    /* we allocated contents in a. So we delete it */
    delete(a[i]);
  }
  /* Why you should NOT do this */
  /*
  for (int i = 0; i < n; i++) {
```

```
  delete(b[i]) ;
  }
  */
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
  Int::setDisplay(true);
  basic();
  test_deque_of_udt();
  test_deque_of_ptr_udt();
  return 1;
}
//EOF
```

VASUDEVAN

12.8 Problem set

Problem 12.8.1.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Two dimensional dynamic array using STL vector

two2darrayusingvector.cpp

```
/*-----------------------------------------------------------------
pattern
-----------------------------------------------------------------*/
static void pattern() {
  const char* s1 =
    "**\n"                  row 0: * *
    "*****\n"               row 1: * * * * *
    "*\n"                   row 2: *
    "***\n"                 row 3: * * *
    ;

  const char* s2 =
    "1\n"
    "12\n"                 row 0: 1
    "123\n"                row 1: 1 2
    "1234\n"               row 2: 1 2 3
    "123\n"                row 3: 1 2 3 4
    "12\n"                 row 4: 1 2 3
    "1\n"                  row 5: 1 2
    ;                      row 6: 1
  {
    vector<vector<char>> a;
    build_2d_array(a, s1); WRITE THESE TWO ROUTINES
    print("pattern", a);    You must store patterns in STL vector only.
  }                         You must print from the stored STL vector
  {                         NO OTHER DATA STRUCTURES CAN BE USED
    vector<vector<char>> a;
    build_2d_array(a, s2);
    print("pattern", a);
  }

}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  pattern();
  return 0;
}
```

Figure 12.5: Building 2D dynamic array using STL vector

VASUDEVAMURTHY

Problem 12.8.2.

VASUDEVAMURTHY

Copyright © Jagadeesh Vasudevamurthy - Do not photocopy.

Snake and Ladder

```
For 20000 rounds, the best: 7 worst: 261
Best move is
1->(4)14->17->22->(28)84->90->(95)75->(80)100 Num dice rolled = 7
Worst move is
1->7->13->14->(16)6->7->13->17->(21)42->(47)26->30->33->35
->(36)44->45->48->50->(51)67->72->74->75->76->81->82->(87)24
->30->33->39->43->45->48->52->53->57->58->(64)60->63->69->(71)91
->(93)73->78->83->88->92->(93)73->78->79->81->(87)24->(29)38->41
->45->(47)26->32->37->38->40->45->(49)11->17->19->22->25->26->31
->35->37->39->40->42->43->(47)26->(29)38->40->46->50->52->57->(62)19->
23->25->30->32->34->39->41->42->46->(49)11->12->15->(21)42->(47)26->
30->33->39->42->46->(47)26->27->(28)84->(87)24->(29)38->40->43->44->
46->50->(56)53->54->55->60->(62)19->22->23->26->27->32->35->37->41->43
->(47)26->(28)84->89->92->(93)73->75->77->81->83->84->(87)24->(28)84->
86->(87)24->25->26->30->32->34->35->39->41->44->46->(51)67->68->70->75->76->
82->(87)24->26->(28)84->90->92->(98)78->84->90->94->(95)75->81->83->(87)24->
27->32->34->40->45->(47)26->27->(29)38->39->42->44->(47)26->32->35->37->41->45
->(47)26->(29)38->42->43->48->(51)67->69->72->74->76->78->84->89->91->94->96
->(98)78->83->(87)24->(29)38->40->44->(49)11->12->14->15->18->23->25->
31->(36)44->(47)26->32->34->37->39->44->48->(49)11->14->20->22->25
->27->(29)38->39->43->(47)26->27->30->34->39->44->(47)26->30->34->35->41
->45->(49)11->13->17->22->(28)84->86->90->(95)75->77->79->83->86->
91->(95)75->81->82->86->88->91->94->100 Num dice rolled = 261
```

Figure 12.6: Snake and Ladder using STL vector

VASUDEVAMURTHY

```cpp
/*-----------------------------------------------------------------
Copyright (c) 2018 Author: Jagadeesh Vasudevamurthy
file: snakeandaddertest.cpp

On linux:
g++ snakeandadder.cpp snakeandaddertest.cpp
valgrind a.out

-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
This file test snakeandadder object
-----------------------------------------------------------------*/

/*-----------------------------------------------------------------
All includes here
-----------------------------------------------------------------*/
#include "snakeandadder.h"

/*-----------------------------------------------------------------
test bed
-----------------------------------------------------------------*/
void testbed() {
  {
    //http://tfortoys.com/product/snake-ladder-paper/#&gid=1&pid=1
    const int NUMROUND = 20000 ;
    const int N = 100;
    const int numl = 9;
    const vector<int> l = { 29, 38, 4, 14, 9, 31, 21, 42, 28,84,36,44,51,67,71,91,80,100 };
    const auto nl = l.size();
    assert(numl * 2 == nl);
    const int nums = 10;
    const vector<int> s = { 98,78,95,75,93,73,87,24,64,60,62,19,56,53,49,11,16,6,47,26 };
    const auto ns = s.size();
    assert(nums * 2 == ns);
    snakeandadder::set_display(true) ;
    if (NUMROUND > 20) {
      snakeandadder::set_display(false) ;
    }
    snakeandadder a(N, l, s, NUMROUND);
  }
}

/*-----------------------------------------------------------------
main
-----------------------------------------------------------------*/
int main() {
```

```
#ifdef _WIN32
  _CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
#endif
  testbed();
  return 0;
}

//EOF


```

VASUDEVAMURTHY