**Data Structure**

**Program**

**Class Notes**

**© Copyright July 2000, 2010**

**by David Burris, Ph.D., CCP, CSP**

**Permission to use this material for personnel educational purposes by students currently registered in my classes is granted. All other rights reserved. (Update 1/4/02)**

**Contents**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Java** |  |  |
|  |  | Singly linked list processing in Java: [class DetentionCell, class CellList, class TestCharList](#DetentionList) | 5 |  |
|  |  | Singly linked deque in Java: [class ListNode, class KlingonList, class KlingonTest](#KlingonList) | 7 |  |
|  |  | Generic Deque in Java, may hold any type of object (via inheritance): class [ListNode, class List, class ListTest](#ListNode) | 11 |  |
|  |  | Simple Bubble Sort in Java: [class simpleBubbleSort](#simpleBubbleSort) | 15 |  |
|  |  | Airline Reservation System Using Java: [Airline](#AirlineJava) | 16 |  |
|  |  | Airline2 Reservation System in Java showing how to extend an existing class using inheritance. [Airline2](#Airline2Java) | 19 |  |
|  |  | Alphabetic Search Tree / Binary Tree: [BinaryTreeNode](#classBinaryTreeNode) | 20 |  |
|  |  |  |  |  |
|  |  | Java binary tree examples and generic sorts |  |  |
|  |  |  |  |  |
|  |  | **Ada** |  |  |
|  |  | Simple linked list in Ada (lsort1): [link1.ada](#link1ada) | 50 |  |
|  |  | Linked list with procedures: [klingon](#klingonada) |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  | Stack by brute force in Ada: [DoStack](#DoStack) | 50 |  |
|  |  | Stack implemented as an Abstract Data Type or simple class in Ada: PrtStack1.adb[PrtStack1Ada](#PrtStack1Ada) | 51 |  |
|  |  | Generic in Ada: [AdaGenerics](#AdaGenerics) | 52 |  |
|  |  | Generic stack in Ada , user specifies the number of items in the stack and the type of the item to be stacked: [Gstack10](#Gstack10) | 53 |  |
|  |  | Sequentially allocated generic circular queue in Ada. The user must specify the data type. The size of the queue is defaulted to 24 if not specified: [Smailbox](#SMailbox) | 54 |  |
|  |  | Dynamic storage allocation in the stack as opposed to the heap. | 56 |  |
|  |  | Generic Arrays | 57 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Example of creating and exporting a programmer defined data type (dates) with I/O routines in Ada: [CompStk1](#CompStk1) | 58 |  |
|  |  | Singly linked list demonstrating use of exported user defined data type (CompStk1) and use of Unchecked\_Deallocation for storage reclaimation in Ada: [UseCmpS1.adb](#UseCmpS1) | 59 |  |
|  |  | Stack implemented as a generic linked list in Ada: [CompStkg](#CompStkg) | 60 |  |
|  |  | Example of programming by "Composition" (bottom-up) as opposed to programming by "Classification" (top-down) better know as the use of inheritance in Ada: [UseCompStkg](#UCmpStkg) |  |  |
|  |  | Single inheritance in Ada (tagged types): [taged1.adb](#tagged1) | 62 |  |
|  |  | Example of programming by "Classification" (top-down) as opposed to programming by "Composition" (bottom-up) in Ada. An abstract stack is created using inheritance and polymorphism: [AbstStck](#AbstStck) |  |  |
|  |  | Creates a rectangle with user defined (generic) data type for length and width. Note that it is frequently desirable to pass methods including I/O routines for programmer defined data types. The following demonstrates how to pass I/O procedures to a generic package. The rectangle may also be used for inheritance if desired. : [GIOEX](#GIOEXads) | 75 |  |
|  |  | Alphabetic Search Tree / binary tree in Ada – recursive build and inorder traversal: [alphatree1.adb](#alphatree1ada) | 79 |  |
|  |  | Alphabetic Search tree with recursive build and iterative inorder traversal: [alphatree2](#alphatree2Ada) | 80 |  |
|  |  | File Organization and Access Methods | 87 |  |
|  |  | Sequential File I/O in Ada | 90 |  |
|  |  | Relative (Direct) File Example (MkDirect.ada) in Ada | 91 |  |
|  |  | Queue implemented as linked list: LinkQ | 92 |  |
|  |  | Sort related records using linked list: LinkSort2 | 94 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  | **“C++”** |  |  |
|  |  | Create a stack for integers in C++: [Stack0.cpp](#stackoCPP) | 68 |  |
|  |  | Templates (generics) in C++ using a stack as an example: [Stack1.cpp](#stack1CPP) | 69 |  |
|  |  | More examples of templates: [stack2.cpp](#stack2CPP) | 70 |  |
|  |  | Create a class for complex numbers in C++. Now use a template class stack to create an object instance that stacks complex numbers: [stack3.cpp](#stack3CPP) | 72 |  |
|  |  | Another stack template example in C++: [stack4.cpp](#stack4CPP) | 74 |  |
|  |  | Alphabet Search Tree (brute force) in CPP: [Btree](#CPPBinarySearchTree) | 81 |  |
|  |  | OO Alphabet Search Tree in CPP: [OOAlphaTree](#OOCPPBinarySearchTree) | 83 |  |
|  |  | OO Template Alphabetic Search Tree: [BSTCTmp](#BSTCTmp) | 85 |  |
|  |  |  |  |  |
|  |  |  |  |  |

**class DetentionCell {**

String Klingon; *// Holds "essence de Klingon!"*

DetentionCell next = null*; // Pointer to next node in list.*

DetentionCell( ) { }

DetentionCell(String c, DetentionCell cell) {Klingon = c; next = cell; }



**}**

**class CellList {**

private DetentionCell head = null;

**public CellList( )** **{ }**

**public CellList(String c)** **{** head = new DetentionCell(c, null); **}**

**public DetentionCell first( ) {** return head; **}** *// Return pointer to first node.*

**public boolean isEmpty() {** return( head == null); **}**

**public DetentionCell last(){** *// Return last node.*

DetentionCell p = head;

while( p != null && p.next != null ) p = p.next; return p;

**}**

**public DetentionCell find(String c*){*** *// Return pointer to specific node.*

for(DetentionCell p = head; p != null; p = p.next)

if( p.Klingon.equalsIgnoreCase(c) ) return p; *//Found.*

return null; *// C not in list.*

**}**

**public void deleteHead(){** if( head != null) head = head.next; }

**public void insertHead(String c){** head = new DetentionCell(c, head ); **}**

**public String toString(){** return toString(head); **}** *// Print whole list*.

**public String toString(DetentionCell p) {** *// From node p on in list.*

String s = "[ ";

while( p != null ){

s = s + p.Klingon; if ( (p = p.next) != null ) s = s + ", ";

}

return( s + " ]");

**}**

**}**

import java.applet.Applet;

import java.awt.Graphics;

**public class TestCharList extends Applet {**

**public void paint(Graphics g) {**

int xAxis = 10, yAxis = 0;

CellList a = new CellList("Joe Klingon");

a.insertHead("Betty Klingon");

a.insertHead("Tally Ho");

g.drawString( "a = " + a.toString(), xAxis, yAxis += 15 );

a.deleteHead();

g.drawString( "a = " + a.toString(), xAxis, yAxis += 15 );

a.insertHead("Phasor Blast");

g.drawString( "a = " + a.toString(), xAxis, yAxis +=15 );

a.insertHead("Totally Cool");

g.drawString( "a = " + a.toString(), xAxis, yAxis +=15 );

DetentionCell pt = a.find("Betty Klingon");

g.drawString("from Betty = " + a.toString( pt ), xAxis , yAxis += 15 );

g.drawString("last node" + a.toString( a.last() ), xAxis, yAxis += 15 );



**}**

**}**

/\*

class TestCharList{

public static void main(String args[]){

CellList a = new CellList("Joe Klingon");

a.insertHead("Betty Klingon");

a.insertHead("Tally Ho");

System.out.println( "a = " + a.toString() );

a.deleteHead();

System.out.println( "a = " + a.toString() );

a.insertHead("Phasor Blast");

System.out.println( "a = " + a.toString() );

a.insertHead("Totally Cool");

DetentionCell pt = a.find("Betty Klingon");

System.out.println( "a = " + a.toString() );

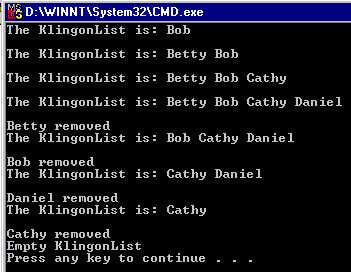
System.out.println("from Betty = " + a.toString( pt ) );

System.out.println("last node" + a.toString( a.last() ) );

}

}

\*/



// Class ListNode and class List definitions

**class ListNode {**

// package access data so class List can access it directly

**String KlingonName;**

**ListNode next;**

// Constructor: Create a ListNode that refers to a Klingon.

**ListNode( String Name ) { this( Name, null ); }**

// Constructor: Create a ListNode that refers to Klingon (Name) and

// to the next ListNode in the list (nextNode).

**ListNode( String Name, ListNode nextNode ) {**

KlingonName = Name; // this node refers to Object o

next = nextNode; // set next to refer to next

**}**

// Return the Object in this node

**String getKlingon() { return KlingonName; }**

// Return the next node

**ListNode getnext() { return next; }**

**}**

// Class List definition

**public class KlingonList {**

**private ListNode firstNode;**

**private ListNode lastNode;**

**private String name;** // String like "list" used in printing

// Constructor: Construct an empty List with s as the name

**public KlingonList( String s ) {**

name = s; firstNode = lastNode = null;

**}**

// Constructor: Construct an empty List with

// "KlingonList" as the name

**public KlingonList() { this( "KlingonList" ); }**

// Insert an Object at the front of the list. If list is empty, firstNode and

// lastNode refer to same Object. Otherwise, firstNode refers to new node.

// Declared to make multithread safe if multiple threads

// attempt to access the list simultaneously.

**public void insertAtFront( String Name ) {**

if ( isEmpty() )

firstNode = lastNode = new ListNode( Name );

else

firstNode = new ListNode( Name, firstNode );

**}**

// Insert an Object at the end of the list. If List is empty, firstNode and lastNode

// refer to same Object. Otherwise, lastNode's next instance variable refers to

// new node.

**public void insertAtBack( String Name ) {**

if ( isEmpty() )

firstNode = new ListNode( Name );

else

lastNode = lastNode.next = new ListNode( Name );

**}**

// Remove the first node from the List.

**public String removeFromFront() {**

String removeName = "";

if ( isEmpty() ) return "";

removeName = firstNode.KlingonName; // retrieve the data

// reset the firstNode and lastNode references

if ( firstNode.next == null )

firstNode = lastNode = null;

else

firstNode = firstNode.next;

return removeName;

**}**

// Remove the last node from the List.

**public String removeFromRear( ){**

String removeName = "";

if ( isEmpty() ) return "";

removeName = lastNode.KlingonName; // retrieve the data

// reset the firstNode and lastNode references

if ( firstNode.equals( lastNode ) )

firstNode = lastNode = null;

else {

ListNode current = firstNode;

while ( current.next != lastNode )

current = current.next;

lastNode = current;

current.next = null;

}

return removeName;

**}**

// Return true if the List is empty

**public boolean isEmpty() { return firstNode == null; }**

// Output the List contents

**public void print() {**

if ( isEmpty() ) {

System.out.println( "Empty " + name ); return;

}

System.out.print( "The " + name + " is: " );

ListNode current = firstNode;

while ( current != null ) {

System.out.print( current.KlingonName + " " );

current = current.next;

}

System.out.println( "\n" );

**}**

**}**

**public class KlingonTest {**

**public static void main( String args[] ) {**

KlingonList List = new KlingonList(); // create the List container

// Create objects to store in the List

Boolean b = new Boolean( true );

String aName = new String("Bob");

// Use the List insert methods

List.insertAtFront( aName ); List.print();

List.insertAtFront( "Betty" ); List.print();

List.insertAtBack( "Cathy" ); List.print();

List.insertAtBack( "Daniel" ); List.print();

// Use the List remove methods

String removedKlingon;

removedKlingon = List.removeFromFront();

System.out.println( removedKlingon + " removed" ); List.print();

removedKlingon = List.removeFromFront();

System.out.println( removedKlingon + " removed" ); List.print();

removedKlingon = List.removeFromRear();

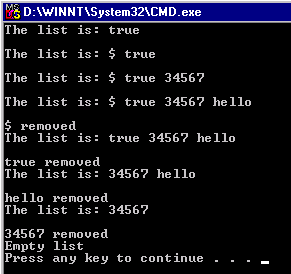
System.out.println( removedKlingon + " removed" ); List.print();

removedKlingon = List.removeFromRear();

System.out.println( removedKlingon + " removed" ); List.print();

**}**

**}**



**class ListNode {**

// package access data so class List can access it directly

**Object data; // *May point to any object!***

**ListNode next;**

// Constructor: Create a ListNode that refers to Object o.

**ListNode( Object o ) { this( o, null ); }**

// Constructor: Create a ListNode that refers to Object o and

// to the next ListNode in the List.

**ListNode( Object o, ListNode nextNode ) {**

data = o; // this node refers to Object o

next = nextNode; // set next to refer to next

**}**

**Object getObject() { return data; }**  // Return the Object in this node

**ListNode getnext() { return next; }**  // Return the next node

**}**

// Class List definition

**public class List {**

**private ListNode firstNode;**

**private ListNode lastNode;**

**private String name;** // String like "list" used in printing

// Constructor: Construct an empty List with s as the name

**public List( String s ) {** name = s; firstNode = lastNode = null; **}**

// Constructor: Construct an empty List with

// "list" as the name

**public List() { this( "list" ); }**

// Insert an Object at the front of the List. // If List is empty, firstNode and lastNode

// refer to same Object. Otherwise, firstNode refers to new node. Declared to make

// multithread safe if multiple threads attempt to access the list simultaneously.

**public synchronized void insertAtFront( Object insertItem ) {**

if ( isEmpty() )

firstNode = lastNode = new ListNode( insertItem );

else

firstNode = new ListNode( insertItem, firstNode );

**}**

// Insert an Object at the end of the List. If List is empty, firstNode and lastNode refer

// to same Object. Otherwise, lastNode's next instance variable refers to new node.

**public synchronized void insertAtBack( Object insertItem ) {**

if ( isEmpty() )

firstNode = new ListNode( insertItem );

else

lastNode = lastNode.next = new ListNode( insertItem );

**}**

// Remove the first node from the List.

**public synchronized Object removeFromFront() {**

Object removeItem = null;

if ( isEmpty() ) return null; // throw new EmptyListException( name );

removeItem = firstNode.data; // retrieve the data

// reset the firstNode and lastNode references

if ( firstNode.next == null )

firstNode = lastNode = null;

else

firstNode = firstNode.next;

return removeItem;

**}**

// Remove the last node from the List.

**public synchronized Object removeFromBack() throws EmptyListException {**

Object removeItem = null;

if ( isEmpty() ) throw new EmptyListException( name );

removeItem = lastNode.data; // retrieve the data

// reset the firstNode and lastNode references

if ( firstNode.equals( lastNode ) )

firstNode = lastNode = null;

else {

ListNode current = firstNode;

while ( current.next != lastNode )

current = current.next;

lastNode = current;

current.next = null;

}

return removeItem;

**}**

// Return true if the List is empty

**public boolean isEmpty() { return firstNode == null; }**

// Output the List contents

**public void print() {**

if ( isEmpty() ) {

System.out.println( "Empty " + name );

return;

}

System.out.print( "The " + name + " is: " );

ListNode current = firstNode;

while ( current != null ) {

System.out.print( current.data.toString() + " " ); current = current.next;

}

System.out.println( "\n" );

}

**}**

// Class ListTest

**public class ListTest {**

**public static void main( String args[] ) {**

List objList = new List(); // create the List container

// Create objects to store in the List

**Boolean b = new Boolean( true );**

**Character c = new Character( '$' );**

**Integer i = new Integer( 34567 );**

**String s = new String( "hello" );**

// Use the List insert methods

**objList.insertAtFront( b ); objList.print();**

**objList.insertAtFront( c ); objList.print();**

**objList.insertAtBack( i ); objList.print();**

**objList.insertAtBack( s ); objList.print();**

// Use the List remove methods

Object removedObj;

try {

removedObj = objList.removeFromFront();

System.out.println( removedObj.toString() + " removed" );

objList.print();

removedObj = objList.removeFromFront();

System.out.println( removedObj.toString() + " removed" );

objList.print();

removedObj = objList.removeFromBack();

System.out.println( removedObj.toString() + " removed" );

objList.print();

removedObj = objList.removeFromBack();

System.out.println( removedObj.toString() + " removed" );

objList.print();

}

catch ( EmptyListException e ) {

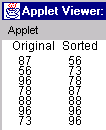
System.err.println( "\n" + e.toString() );

}

}

}

import java.applet.Applet;



import java.awt.Graphics;

**public class simpleBubbleSort extends Applet {**

**public void paint( Graphics g ){**

**int grades[ ]** = { 87, 56, 96, 78, 88, 96, 73 };

print( **g**, "Original", **grades**, 10, 10 );

bubbleSort( **grades** );

print( **g**, "Sorted", **grades**, 60, 10 );

**}**

**public void print( Graphics g, String message, int array[ ], int row, int col ){**

g.drawString( message, row, col);

row += 6; col += 15;

for(int pt = 0; pt < **array.length**; pt++) {

g.drawString( String.valueOf( **array[pt]** ), row, col );

col += 10;

}

**}**

**private void bubbleSort( int grades[] ){** *// Ascending order.*

boolean switched = true; *// Stop if nothing is switched during a pass.*

int temp; *// Maximum length – 1 passes.*

for( int pass = grades.length; (pass > 0) && switched; pass--){

switched = false; *// At top nothing switched.*

for(int k = 0; k < (pass - 1); k++){

if(grades[k] > grades[k+1] ){

temp = grades[k];

grades[k] = grades[k+1]; grades[k+1] = temp;

switched = true;

}

}

}

**}**

**}**

import java.awt.\*; // In file Airline.java

import java.applet.Applet;

import java.util.\*;

class Reservation{

int flightNumber;

String dateOfTravel;

String seatNumber;

public Reservation(){

flightNumber = 0; dateOfTravel = "unscheduled"; seatNumber = "no seat";

}

public Reservation(int flightNum, String travDate, String seat){

System.out.println(" ");

System.out.println("Reservation recorded!");

flightNumber = flightNum; dateOfTravel = travDate;

seatNumber = seat;

}

public String toString(){

return ("Flight " + flightNumber + " " +

dateOfTravel + " " + seatNumber);

}

}

class BasicReservation extends Reservation{

public BasicReservation(){ super(); }

public BasicReservation(int flightNum, String travDate, String seat){

flightNumber = flightNum; dateOfTravel = travDate;

seatNumber = seat;

// super(flighNumber, travDate, seat); would be more object oriented.

}

public String toString(){return "Basic: " + super.toString();}

}

class NiceReservation extends Reservation{

String seatType;

String meal;

public NiceReservation(){

super(); seatType = "plain"; meal = "yucky";

}

public NiceReservation(int flightNum, String travDate, String seat,

String seatLoc, String mealOrdered){

super(flightNum, travDate, seat);

seatType = seatLoc; meal = mealOrdered;

}

public String toString(){

return "Nice " + super.toString()+ " " +

seatType + " " + meal;

}

}

class PoshReservation extends NiceReservation{

String Destination;

PoshReservation(int flightNum, String travDate, String seat,

String seatLoc, String mealOrdered, String goingTo){

super(flightNum, travDate, seat, seatLoc, mealOrdered);

Destination = goingTo;

}

public String toString(){

return "Posh " + super.toString() + " " + Destination;

}

}

**public class Airline extends Applet{**

Reservation rev1; BasicReservation basic1; NiceReservation nice1; PoshReservation posh1;

**public void init(){**  // Compile time binding.

rev1 = new Reservation( 1124, "1/15/98", "A23");

basic1 = new BasicReservation( 1125, "1/23/98", "C12");

nice1 = new NiceReservation(2236, "1/23/99", "A4", "Business Class", "Fish");

posh1 = new PoshReservation(3724, "1/23/98", "A3", "First Class", "Steak", "Grand Canyon");

**}**

**public void paint(Graphics g){**  // Compile time binding.

int xPos = 10, yPos = 10;

g.drawString( rev1.toString(), xPos, yPos += 15);

g.drawString( basic1.toString(), xPos, yPos += 15 );

g.drawString( nice1.toString(), xPos, yPos += 15 );

g.drawString( posh1.toString(), xPos, yPos += 15);

Reservation rev[ ] = new Reservation[5];

rev[0] = rev1;

rev[1] = basic1;

rev[2] = nice1;

rev[3] = posh1;

rev[4] = basic1;

g.drawString( "Start polymorphic calls: ", xPos, yPos +=30 );

for(int i = 0; i < 5; i++) {

g.drawString( rev[i].toString(), xPos, yPos += 15); // Polymorphic call.

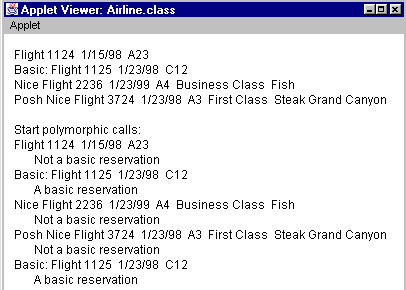
// An object always knows its class.

if (rev[i] instanceof BasicReservation)

g.drawString("A basic reservation", xPos + 20, yPos += 15);

else

g.drawString("Not a basic reservation", xPos + 20, yPos += 15);



}

}

}

import java.awt.\*;

import java.applet.Applet;

import java.util.\*;

public class Airline2 extends Applet{

Reservation rev1;

BasicReservation basic1;

NiceReservation nice1;

PoshReservation posh1;

public void init(){ // Compile time binding.

rev1 = new Reservation( 1124, "1/15/98", "A23");

basic1 = new BasicReservation( 1125, "1/23/98", "C12");

nice1 = new NiceReservation(2236, "1/23/99", "A4",

"Business Class", "Fish");

posh1 = new PoshReservation(3724, "1/23/98", "A3", "First Class",

"Steak", "Grand Canyon");

Reservation rev[] = new Reservation[5];

rev[0] = rev1; rev[1] = basic1; rev[2] = nice1; rev[3] = posh1;

rev[4] = basic1;

System.out.println("Polymorphic method invocation ");

for(int i = 0; i < 5; i++) {

System.out.println("rev[" + i + "] = " + rev[i].toString());

**// An object always knows its class.**

**if (rev[i] instanceof BasicReservation)**

**System.out.println("A basic reservation");**

**else**

**System.out.println("Not a basic reservation");**

}

System.out.println(" ");

System.out.println("After changing order of array contents.");

rev[0] = posh1; rev[1] = nice1; rev[2] = basic1; rev[4] = rev1;

for(int i = 0; i < 5; i++) {

System.out.println("rev[" + i + "] = " + rev[i].toString());

// An object always knows its class.

if (rev[i] instanceof BasicReservation)

System.out.println("A basic reservation");

else

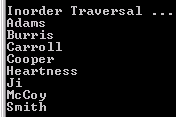
System.out.println("Not a basic reservation");

}

}

}

**class BinaryTreeNode {**



**BinaryTreeNode left, right;**

**private String name;**

**public BinaryTreeNode( String nam )**

**{** name = new String(nam); left = right = null; **}**

**public synchronized void insertBinaryTreeNode( String nam ) {**

if( nam.compareTo( name) < 0 ) { // name is < name

if( left == null )

left = new BinaryTreeNode( nam );

else

left.insertBinaryTreeNode( nam );

}

else if( nam.compareTo( name ) >= 0 ) { // nam >= name

if( right == null )

right = new BinaryTreeNode( nam );

else

right.insertBinaryTreeNode( nam );

}

**}**

**public synchronized String getName( ) { return name; }**

**}**

**public class BinaryTree {**

**private BinaryTreeNode root;**

**public BinaryTree( )** **{** root = null; **}**

**public synchronized void insertNode( String nam ) {**

if( root == null )

root = new BinaryTreeNode( nam );

else

root.insertBinaryTreeNode( nam );

**}**

**public void preorderTraversal( ) {** preorder( root ); **}**

**private void preorder( BinaryTreeNode nodePt ){**

if( nodePt == null ) return;

System.out.println( nodePt.getName() );

preorder( nodePt.left );

preorder( nodePt.right );

**}**

**public void inorderTraversal( ) {** inorder( root ); **}**

**private void inorder( BinaryTreeNode nodePt ){**

if( nodePt == null ) return;

inorder( nodePt.left );

System.out.println( nodePt.getName() );

inorder( nodePt.right );

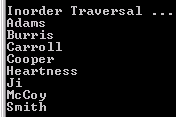
**}**

**public void postorderTraversal( ) {** postorder( root ); **}**

**private void postorder( BinaryTreeNode nodePt ){**

if( nodePt == null ) return;

postorder( nodePt.left );



postorder( nodePt.right );

System.out.println( nodePt.getName() );

**}**

**}**

**public class BinaryTreeTest {**

**public static void main( String args[ ] ) {**

**BinaryTree tree** = new **BinaryTree()**;

String faculty;

faculty = "Burris"; **tree.insertNode( faculty );**

**tree.insertNode ("Adams" ); tree.insertNode( "Smith" );**

**tree.insertNode( "Cooper ); tree.insertNode( "McCoy" );**

**tree.insertNode( "Carroll" ); tree.insertNode( "Ji" );**

**tree.insertNode( "Heartness" );**

System.out.println("\nPreorder Traversal ...");

**tree.preorderTraversal();**

System.out.println("\nInorder Traversal ... ");

**tree.inorderTraversal();**

System.out.println("\nPost Order Traversal ... ");

**tree.postorderTraversal();**

**}**

**}**

File Organization

And

Access Methods

Organization:

1. Sequential
2. Indexed
3. Relative

Access Methods:

1. Sequential
2. Random
3. Dynamic

--file: RelativeFiles.doc

Random Access Methods

Records are stored and retrieved on the basis of a predictable relationship between the key of the desired record and the location where the record is stored.

Methods

1. Direct: The user supplies either the direct address on the storage device of the desired record (e.g., on a disk the cylinder, track, and sector) or a virtual address that can be transformed easily to the actual address either by the system software or device hardware (e.g., a relative record number for conversion).
2. Dictionary Look-up.
3. Hashing or Calculation.

Relative Files

Traditional – Fixed Length Records

All space for file is typically allocated as a contiguous unit at the time the file is created.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  | Betty | Adam |  | Sue |  | Tom | ••• |
| Record  Number | 0 | 1 | 2 | 3 | 4 | 5 |  |
| Device  Address | 0 | 50 | 100 | 150 | 200 | 250 |  |

Loc[RECJ ] = Base + Offset

Offset = J \* number of characters in record

**Example:** Find relative record 3 assuming 50 characters per record starting at a base address of zero on the storage device.

**LOC[ REC3 ] = 0 + 3 \* 50 = 150**.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

**Relative files may be based on a table lookup.** This is particularly attractive for:

1. variable length records,
2. in situations where dynamic storage allocation is desirable.

Storage Device Address

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Index |  |  |  |  |
|  | 0 | 336 |  | Adam | 105 |  |
|  | 1 | 105 |  |  |  |  |
|  | 2 |  |  |  |  |  |
|  | 3 | 768 |  | Betty | 336 |  |
|  | 4 |  |  |  |  |  |
|  | 5 |  |  |  |  |  |
|  | ••• |  |  | Sue | 768 |  |
|  |  |  |  |  |  |  |

**Sequential File Processing in Java**

// Create sequential file. Based on Deitel & Deitel, 2nd Ed., Ch. 15. Assume an employee

// record consists of an identification number, name, department, and payrate.

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

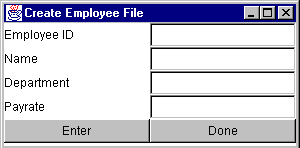
**public class CreateSeqEmployeeFile extends Frame implements ActionListener {**

// TextFields where user enters account number, name, department and payrate.

private TextField empIDField, nameField, departmentField, payrateField;

private Button enter, // send record to file

done; // quit program



**private DataOutputStream output;**

**public CreateSeqEmployeeFile() {**

super( "Create Employee File" );

// Open the file

try {

**output = new DataOutputStream(**

**new FileOutputStream( "employee.dat" ) );**

}

catch ( IOException e ) {

System.err.println( "File not opened properly\n" + e.toString() );

System.exit( 1 );

}

setSize( 300, 150 ); setLayout( new GridLayout( 5, 2 ) );

// create the components of the Frame

add( new Label( "Employee ID" ) );

empIDField = new TextField(); add( empIDField );

add( new Label( "Name" ) );

nameField = new TextField( 20 ); add( nameField );

add( new Label( "Department" ) );

departmentField = new TextField( 20 ); add( departmentField );

add( new Label( "Payrate" ) );

payrateField = new TextField( 20 ); add( payrateField );

enter = new Button( "Enter" ); enter.addActionListener( this ); add( enter );

done = new Button( "Done" ); done.addActionListener( this ); add( done );

setVisible( true );

**}**

**public void addRecord() {**

int empIDNumber = 0;

Double payrate;

if ( ! empIDField.getText().equals( "" ) ) {

try { // output the values to the file

empIDNumber = Integer.parseInt( empIDField.getText() );

if ( empIDNumber > 0 ) {

**output.writeInt( empIDNumber );**

**output.writeUTF( nameField.getText() );**

**output.writeUTF( departmentField.getText() );**

payrate = new Double ( payrateField.getText() );

**output.writeDouble( payrate.doubleValue() );**

}

// clear the TextFields

empIDField.setText( "" ); nameField.setText( "" );

departmentField.setText( "" ); payrateField.setText( "" );

}

catch ( NumberFormatException nfe ) {

System.err.println( "You must enter an integer employee ID!" );

}

catch ( IOException io ) {

System.err.println(

"Error during write to file\n" + io.toString() );

System.exit( 1 );

}

}

}

public void actionPerformed( ActionEvent e ) {

addRecord();

if ( e.getSource() == done ) {

try {

output.close();

}

catch ( IOException io ) {

System.err.println( "File not closed properly\n" + io.toString() );

}

System.exit( 0 );

}

**}**

**public static void main( String args[] ) {**

new CreateSeqEmployeeFile();

**}**

**}**

// This program reads a sequential file created by

// CreateSeqEmpFile.java and displays each record in text fields.

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

**public class ReadSeqEmpFile extends Frame implements ActionListener {**

private TextField empIDField, nameField, departmentField, payrateField;

private Button next, // get next record in file

done; // quit program

**private DataInputStream input;**

**public ReadSeqEmpFile() {**

super( "Read Client File" );

try { // Open the file

**input = new DataInputStream(**

**new FileInputStream( "employee.dat" ) );**

}

catch ( IOException e ) {

System.err.println( "File not opened properly\n" + e.toString() );

System.exit( 1 );

}

setSize( 300, 150 ); setLayout( new GridLayout( 5, 2 ) );

// create the components of the Frame

add( new Label( "Employee ID" ) ); empIDField = new TextField();

empIDField.setEditable( false ); add( empIDField );

add( new Label( "Name" ) ); nameField = new TextField( 20 );

nameField.setEditable( false ); add( nameField );

add( new Label( "Department" ) ); departmentField = new TextField( 20 );

departmentField.setEditable( false ); add( departmentField );

add( new Label( "Payrate" ) ); payrateField = new TextField( 20 );

payrateField.setEditable( false ); add( payrateField );

next = new Button( "Next" ); next.addActionListener( this ); add( next );

done = new Button( "Done" ); done.addActionListener( this ); add( done );

setVisible( true );

**}**

**public void actionPerformed( ActionEvent e ) {**

if ( e.getSource() == next )

readRecord();

else

**closeFile();**

**}**

**public void readRecord() {**

int empID;

String name, department;

double payrate;

try { // input the values from the file

**empID = input.readInt();**

**name = input.readUTF();**

**department = input.readUTF();**

**payrate = input.readDouble();**

empIDField.setText( String.valueOf( empID ) );

nameField.setText( name );

departmentField.setText( department );

payrateField.setText( String.valueOf( payrate ) );

}

catch ( **EOFException eof** ) {

**closeFile( );**

}

catch ( IOException e ) {

System.err.println( "Error during read from file\n" + e.toString() );

System.exit( 1 );

}

**}**

**private void closeFile(){**

try {

**input.close();**

System.exit( 0 );

}

catch ( IOException e ) {

System.err.println( "Error closing file\n" + e.toString() );

System.exit( 1 );

}

}

**public static void main( String args[] ){**

new ReadSeqEmpFile();

}

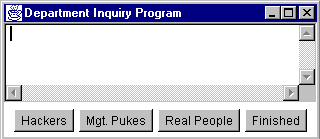
**}**

// Reads employee file created by CreateSeqEmpFile

// and reports by department a list of managers (Mgt),

// CS personnel (Hackers), and other employees (Real People).

import java.io.\*;



import java.awt.\*;

import java.awt.event.\*;

import java.text.DecimalFormat;

**public class DeptInquery extends Frame**

**implements ActionListener {**

// window components

private TextArea recordDisplay;

private Button finished, csEmp, mgtEmp, otherEmp;

private Panel buttonPanel;

private RandomAccessFile input;

private String accountType;

public DeptInquery() {

super( "Department Inquiry Program" );

// Open the file

try {

**input = new RandomAccessFile( "employee.dat", "r" );**

}

catch ( IOException e ) {

System.err.println( e.toString() );

System.exit( 1 );

}

setSize( 400, 150 );

// create Frame

buttonPanel = new Panel();

csEmp = new Button( "Hackers" );

csEmp.addActionListener( this );

buttonPanel.add( csEmp );

mgtEmp = new Button( "Mgt. Pukes" );

mgtEmp.addActionListener( this );

buttonPanel.add( mgtEmp );

otherEmp = new Button( "Real People" );

otherEmp.addActionListener( this );

buttonPanel.add( otherEmp );

finished = new Button( "Finished" );

finished.addActionListener( this );

buttonPanel.add( finished );

recordDisplay = new TextArea( 4, 40 );

// add the components to the Frame

add( recordDisplay, BorderLayout.NORTH );

add( buttonPanel, BorderLayout.SOUTH );

setVisible( true );

**}**

**public void actionPerformed( ActionEvent e ) {**

if ( e.getSource() != finished ) {

accountType = e.getActionCommand();

readRecords();

}

else { // Close the file

try {

**input.close();**

System.exit( 0 );

}

catch ( IOException ioe ) {

System.err.println( "File not closed properly\n" + ioe.toString() );

System.exit( 1 );

}

}

**}**

**public void readRecords() {**

int empID;

String name, department;

double payrate;

DecimalFormat twoDigits = new DecimalFormat( "0.00" );

// input the values from the file

try { // to catch IOException

try { // to catch EOFException

recordDisplay.setText( "Requested employees are:\n" );

while ( true ) {

empID = input.readInt();

name = input.readUTF();

department = input.readUTF();

payrate = input.readDouble();

if ( shouldDisplay( department ) )

recordDisplay.append( department + "\t" +

name + "\t" + empID + "\t" +

twoDigits.format( payrate ) + "\n" );

}

}

catch ( EOFException eof ) {

**input.seek( 0 );**

}

}

catch ( IOException e ) {

System.err.println( "Error during read from file\n" +

e.toString() );

System.exit( 1 );

}

**}**

**public boolean shouldDisplay( String department ) {**

if ( accountType.equals( "Hackers" ) &&

department.equals("CS") )

return true;

else if ( accountType.equals( "Mgt. Pukes" ) &&

department.equals( "Mgt" ) )

return true;

else if ( accountType.equals( "Real People" ) &&

(!department.equals( "CS" )) &&

(!department.equals( "Mgt" ) ) )

return true;

return false;

}

// Instantiate a DepartmentInquery object and start the program

**public static void main( String args[] ) {**

new DeptInquery();

}

**}**

**Creating and processing Relative Files in Java**

// Employee record class for the RandomAccessFile programs.

import java.io.\*;

**public class EmpRecord {**

Employee record format (92 bytes):

|  |  |  |  |
| --- | --- | --- | --- |
| EmpID | EmpName | EmpDept | Payrate |
| integer | Char[20] | Char[20] | double |

Integer => 4 bytes, char => 2 bytes, double => 8 bytes

private int empID;

private String empName;

private String empDepartment;

private double payrate;

// Read a record from the specified RandomAccessFile

**public void read( RandomAccessFile file ) throws IOException{**

**empID = file.readInt();**

char name[] = new char[ 20 ];

**for ( int i = 0; i < name.length; i++ ) name[ i ] = file.readChar();**

**empName = new String( name );**

char department[] = new char[ 20 ];

**for ( int i = 0; i < department.length; i++ ) department[ i ] = file.readChar();**

**empDepartment = new String( department );**

**payrate = file.readDouble();**

**}**

// Write a record to the specified RandomAccessFile

**public void write( RandomAccessFile file ) throws IOException {**

StringBuffer buf;

**file.writeInt( empID );**

if ( empName != null )

buf = new StringBuffer( empName );

else

buf = new StringBuffer( 20 );

buf.setLength( 20 );

**file.writeChars( buf.toString() );**

if ( empDepartment != null )

buf = new StringBuffer( empDepartment );

else

buf = new StringBuffer( 20 );

buf.setLength( 20 );

**file.writeChars( buf.toString() );**

**file.writeDouble( payrate );**

**}**

// Access functions.

**public void setEmpID( int eid ) { empID = eid; }**

**public int getEmpID() { return empID; }**

**public void setEmpName( String en ) { empName = en; }**

**public String getEmpName() { return empName; }**

**public void setEmpDepartment( String dept ) { empDepartment = dept; }**

**public String getEmpDepartment() { return empDepartment; }**

**public void setPayrate( double pr ) { payrate = pr; }**

**public double getPayrate() { return payrate; }**

// NOTE: This method contains a hard coded value for the

// size of a record of information. 4-bytes for an integer, 8-bytes for double, and

// 2 for each unicode char implies 4 + 40 + 40 + 8 = 92.

**public static int size() { return 92; }**

**}**

// Fig. 15.11: CreateRandFile.java

// This program creates a random access file sequentially by writing 20 empty records to disk.

import java.io.\*;

**public class CreateRandEmpFile {**

private EmpRecord blank;

private RandomAccessFile file;

**public CreateRandEmpFile() {**

blank = new EmpRecord();

try {

**file = new RandomAccessFile( "EmployeeMaster.dat", "rw" );**

for ( int i = 0; i < 20; i++ )

**blank.write( file );**

}

catch( IOException e ) {

System.err.println( "File not opened properly\n" + e.toString() ); System.exit( 1 );

}

// Java is supposed to close all files when an application terminates.

// It is better to explicitly close each file. “file.close();”

**}**

**public static void main( String args[ ] ) {**

CreateRandEmpFile empMaster = new CreateRandEmpFile();

**}**

**}**

import java.io.\*;

import java.awt.\*;

import java.awt.event.\*;

**public class WriteRandEmpFile extends Frame implements ActionListener {**

// TextFields where user enters account number, first name,

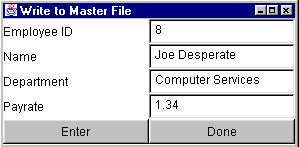
// last name and balance.

private TextField empIDField, nameField, departmentField, payrateField;

private Button enter, // send record to file

done; // quit program

**private RandomAccessFile output;** // file for output



**private EmpRecord data;**

**public WriteRandEmpFile() {**

super( "Write to Master File" );

**data = new EmpRecord();**

try { // Open the file with "read" and "write" access.

**output = new RandomAccessFile( "empMaster.dat", "rw" );**

}

catch ( IOException e ) {

System.err.println( e.toString() ); System.exit( 1 );

}

setSize( 300, 150 ); setLayout( new GridLayout( 5, 2 ) );

add( new Label( "Employee ID" ) ); empIDField = new TextField();

add( empIDField );

add( new Label( "Name" ) ); nameField = new TextField( 20 ); add( nameField );

add( new Label( "Department" ) );

departmentField = new TextField( 20 ); add( departmentField );

add( new Label( "Payrate" ) ); payrateField = new TextField( 20 );

add( payrateField );

enter = new Button( "Enter" ); enter.addActionListener( this ); add( enter );

done = new Button( "Done" ); done.addActionListener( this ); add( done );

setVisible( true );

**}**

**public void addRecord() {**

int empIDNumber = 0;

Double payrate;

if ( ! empIDField.getText().equals( "" ) ) {

try { // output the values to the file

empIDNumber = Integer.parseInt( empIDField.getText() );

if ( empIDNumber > 0 && empIDNumber <= 20 ) {

**data.setEmpID( empIDNumber );**

**data.setEmpName( nameField.getText() );**

**data.setEmpDepartment( departmentField.getText() );**

**payrate = new Double ( payrateField.getText() );**

**data.setPayrate( payrate.doubleValue() );**

**output.seek( (long) ( empIDNumber-1 ) \* EmpRecord.size() );**

**data.write( output );**

}

empIDField.setText( "" ); nameField.setText( "" ); // clear the TextFields

departmentField.setText( "" ); payrateField.setText( "" );

}

catch ( NumberFormatException nfe ) {

System.err.println( "You must enter an integer employee ID number" );

}

catch ( IOException io ) {

System.err.println( "Error during write to file\n" + io.toString() ); System.exit( 1 );

}

}

**}**

**public void actionPerformed( ActionEvent e ) {**

addRecord();

if ( e.getSource() == done ) {

try {

output.close();

}

catch ( IOException io ) {

System.err.println( "File not closed properly\n" +io.toString() );

}

System.exit( 0 );

}

}

// Instantiate a WriteRandomFile object and start the program

**public static void main( String args[] ) {** new WriteRandEmpFile(); **}**

**}**

// This program reads the random access employee master file sequentially and

// displays the contents one record at a time in text fields.

import java.io.\*;

import java.awt.\*;

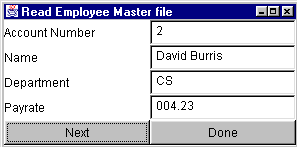
import java.awt.event.\*;

import java.text.DecimalFormat;

**public class ReadRandEmpFile extends Frame implements ActionListener {**

private TextField empIDField, nameField, departmentField, payrateField;

private Button next, done;



**private RandomAccessFile input;**

**private EmpRecord data;**

**public ReadRandEmpFile() {**

super( "Read Employee Master file" );

try { // Read-only access.

**input = new RandomAccessFile( "empMaster.dat", "r" );**

}

catch ( IOException e ) {

System.err.println( e.toString() ); System.exit( 1 );

}

**data = new EmpRecord();**

setSize( 300, 150 ); setLayout( new GridLayout( 5, 2 ) );

add( new Label( "Account Number" ) ); empIDField = new TextField();

empIDField.setEditable( false ); add( empIDField );

add( new Label( "Name" ) ); nameField = new TextField( 20 );

nameField.setEditable( false ); add( nameField );

add( new Label( "Department" ) ); departmentField = new TextField( 20 );

departmentField.setEditable( false ); add( departmentField );

add( new Label( "Payrate" ) ); payrateField = new TextField( 20 );

payrateField.setEditable( false ); add( payrateField );

next = new Button( "Next" ); next.addActionListener( this ); add( next );

done = new Button( "Done" ); done.addActionListener( this ); add( done );

setVisible( true );

**}**

**public void actionPerformed( ActionEvent e ) {**

if ( e.getSource() == next )

**readRecord();**

else

**closeFile();**

**}**

**public void readRecord() {**

DecimalFormat twoDigits = new DecimalFormat( "0.00" );

// read a record and display

try {

do {

**data.read( input );**

} while ( data.getEmpID() == 0 );

**empIDField.setText( String.valueOf( data.getEmpID() ) );**

**nameField.setText( data.getEmpName() );**

**departmentField.setText( data.getEmpDepartment() );**

**payrateField.setText( String.valueOf( twoDigits.format( data.getPayrate() ) ) );**

}

catch ( EOFException eof ) {

closeFile();

}

catch ( IOException e ) {

System.err.println( "Error during read from file\n" + e.toString() ); System.exit( 1 );

}

**}**

**private void closeFile( {**

try {

**input.close();**

System.exit( 0 );

}

catch ( IOException e ) {

System.err.println( "Error closing file\n" + e.toString() ); System.exit( 1 );

}

**}**

// Instantiate a ReadRandomFile object and start the program

**public static void main( String args[ ] ) {**

new ReadRandEmpFile();

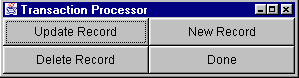
**}**

**}**

// This program reads a random access file,

// updates data already written to the file, creates new

// data to be placed in the file, and deletes data



// already in the file.

import java.awt.\*;

import java.awt.event.\*;

import java.io.\*;

import java.text.DecimalFormat;

**public class TransactEmpMas extends Frame implements ActionListener {**

// Application window components

private Button update, // update record

newRecord, // add new record

delete, // delete record

done; // quit program

private UpdateDialog updateDialog; // record update

private NewDialog newDialog; // add new records

private DeleteDialog deleteDialog; // delete records

private RandomAccessFile file;

private EmpRecord data;

**public TransactEmpMas() {**

super( "Transaction Processor" );

// Open the file “Read & Write access ‘ only.

try {

**file = new RandomAccessFile( "empMaster.dat", "rw" );**

}

catch ( IOException ioe ) {

System.err.println( ioe.toString() ); System.exit( 1 );

}

**data = new EmpRecord();**

setSize( 300, 80 ); setLayout( new GridLayout( 2, 2 ) );

update = new Button( "Update Record" );

update.addActionListener( this ); add( update );

newRecord = new Button( "New Record" );

newRecord.addActionListener( this ); add( newRecord );

delete = new Button( "Delete Record" );

delete.addActionListener( this ); add( delete );

done = new Button( "Done" );

done.addActionListener( this ); add( done );

setVisible( true ); // show the Frame

// Create dialog boxes

**updateDialog = new UpdateDialog( this, file );**

**newDialog = new NewDialog( this, file );**

**deleteDialog = new DeleteDialog( this, file );**

**}**

// Process actions

**public void actionPerformed( ActionEvent e ) {**

**if ( e.getSource() == update )**

**updateDialog.setVisible( true );**

**else if ( e.getSource() == newRecord )**

**newDialog.setVisible( true );**

**else if ( e.getSource() == delete )**

**deleteDialog.setVisible( true );**

**else**

**closeFile( );**

**}**

**public void closeFile() {**

try {

**file.close(); System.exit( 0 );**

}

catch ( IOException ioe ) {

System.err.println( ioe.toString( ) ); System.exit( 1 );

}

**}**

**public static void main( String args[ ] ) {**

new TransactEmpMas();

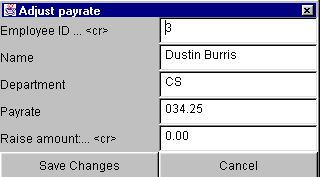
**}**

**}**

**class UpdateDialog extends Dialog implements ActionListener {**

private TextField empIDField, nameField, departmentField,

payrateField, raiseField;



1. Enter empID and hit return.
2. Enter raise and hit return.
3. “Save” or “Cancel” to complete.

private Button cancel, save;

private RandomAccessFile file;

private EmpRecord data;

private int empIDNumber;

**public UpdateDialog( TransactEmpMas t,**

**RandomAccessFile f ) {**

super( t, "Adjust payrate", true );

**file = f;**

**data = new EmpRecord();**

setSize( 320, 180 ); setLayout( new GridLayout( 6, 2 ) );

add( new Label( "Employee ID ... <cr>" ) ); empIDField = new TextField();

empIDField.addActionListener( this ); add( empIDField );

add( new Label( "Name" ) ); nameField = new TextField(); add( nameField );

add( new Label( "Department" ) ); departmentField = new TextField();

add( departmentField );

add( new Label( "Payrate" ) ); payrateField = new TextField(); add( payrateField );

add( new Label( "Raise amount:... <cr>" ) ); raiseField = new TextField();

raiseField.addActionListener( this ); add( raiseField );

save = new Button( "Save Changes" ); save.addActionListener( this ); add( save );

cancel = new Button( "Cancel" ); cancel.addActionListener( this ); add( cancel );

**}**

**public void actionPerformed( ActionEvent e ) {**

if ( e.getSource() == empIDField ) {

**empIDNumber = Integer.parseInt( empIDField.getText() );**

**readAccountInformation();**

}

else if ( e.getSource() == save ) {

**empIDNumber = Integer.parseInt( empIDField.getText() );**

**updateAccountInformation();**

}

else if ( e.getSource() == cancel ) {

**setVisible( false ); clear();**

}

else if ( e.getSource() == raiseField ) {

DecimalFormat twoDigits = new DecimalFormat( "0.00" );

**Double transactionAmount = Double.valueOf( raiseField.getText() );**

**data.setPayrate( data.getPayrate() + transactionAmount.doubleValue() );**

**payrateField.setText( twoDigits.format( data.getPayrate() ) );**

}

}

**private void readAccountInformation() {**

DecimalFormat twoDigits = new DecimalFormat( "0.00" );

if ( empIDNumber < 1 || empIDNumber > 20 ) {

**empIDField.setText( "Invalid account" ); return;**

}

try {

**file.seek( ( empIDNumber - 1 ) \* EmpRecord.size() );**

**data.read( file );**

if ( data.getEmpID() != 0 ) {

**empIDField.setText( String.valueOf( data.getEmpID() ) );**

**nameField.setText( data.getEmpName() );**

**departmentField.setText( data.getEmpDepartment() );**

**payrateField.setText( twoDigits.format( data.getPayrate() ) );**

**raiseField.setText( "0.00" );**

}

else

**empIDField.setText( String.valueOf( empIDNumber ) + " does not exist" );**

}

catch ( IOException ioe ) {

**empIDField.setText( "Error reading file" );**

}

**}**

**private void updateAccountInformation() {**

try {

**file.seek( ( empIDNumber - 1 ) \* EmpRecord.size() );**

**data.write( file );**

}

catch ( IOException ioe ) {

empIDField.setText( "Error writing file" ); return;

}

setVisible( false ); clear();

**}**

**private void clear() {**

empIDField.setText( "" ); nameField.setText( "" );

departmentField.setText( "" ); payrateField.setText( "" );

raiseField.setText( "" );

**}**

**}**

**class NewDialog extends Dialog implements ActionListener {**

private RandomAccessFile file;

private TextField empIDField, nameField, departmentField, payrateField;

private Button cancel, save;

private EmpRecord data;

private int empIDNumber;

**public NewDialog( TransactEmpMas t, RandomAccessFile f ) {**

super( t, "New Record", true );

file = f;

data = new EmpRecord();

setSize( 300, 150 ); setLayout( new GridLayout( 5, 2 ) );

add( new Label( "Account" ) ); empIDField = new TextField();

empIDField.addActionListener( this ); add( empIDField );

add( new Label( "Name" ) ); nameField = new TextField(); add( nameField );

add( new Label( "Department" ) ); departmentField = new TextField();

add( departmentField );

add( new Label( "Payrate" ) ); payrateField = new TextField(); add( payrateField );

save = new Button( "Save Changes" ); save.addActionListener( this ); add( save );

cancel = new Button( "Cancel" ); cancel.addActionListener( this ); add( cancel );

**}**

**public void actionPerformed( ActionEvent e ) {**

if ( e.getSource() == empIDField )

**checkIfAccountExists();**

else if ( e.getSource() == save )

**addRecord();**

else if ( e.getSource() == cancel ) {

**setVisible( false ); clear();**

}

**}**

**private void checkIfAccountExists() {**

empIDNumber = Integer.parseInt( empIDField.getText() );

if ( empIDNumber < 1 || empIDNumber > 20 ) {

**empIDField.setText( "Invalid account" ); return;**

}

try {

**file.seek( ( empIDNumber - 1 ) \* EmpRecord.size() );**

**data.read( file );**

}

catch ( IOException ioe ) {

**empIDField.setText( "Error reading file" );**

}

if ( data.getEmpID() == 0 ) {

nameField.setText( "Enter name" );

departmentField.setText( "Enter department" );

payrateField.setText( "Enter payrate" );

}

else {

empIDField.setText( data.getEmpID() + " already exists" );

nameField.setText( "" ); departmentField.setText( "" ); payrateField.setText( "" );

}

**}**

**private void addRecord() {**

try {

data.setEmpID( empIDNumber );

data.setEmpName( nameField.getText() );

data.setEmpDepartment( departmentField.getText() );

data.setPayrate( ( new Double ( payrateField.getText() ) ).doubleValue() );

file.seek( ( empIDNumber - 1 ) \* EmpRecord.size() );

**data.write( file );**

}

catch ( IOException ioe ) {

empIDField.setText( "Error writing file" ); return;

}

setVisible( false );

clear();

**}**

**private void clear() {**

empIDField.setText( "" ); nameField.setText( "" );

departmentField.setText( "" ); payrateField.setText( "" );

**}**

**}**

**class DeleteDialog extends Dialog implements ActionListener {**

private RandomAccessFile file; private TextField empIDField;

private Button cancel, delete; private EmpRecord data; private int empIDNumber;

**public DeleteDialog( TransactEmpMas t, RandomAccessFile f ){**

super( t, "Delete Record", true ); file = f; data = new EmpRecord();

setSize( 300, 80 ); setLayout( new GridLayout( 2, 2 ) );

add( new Label( "Employee ID" ) ); empIDField = new TextField();

empIDField.addActionListener( this ); add( empIDField );

delete = new Button( "Delete Record" ); delete.addActionListener( this ); add( delete );

cancel = new Button( "Cancel" ); cancel.addActionListener( this ); add( cancel );

**}**

**public void actionPerformed( ActionEvent e ) {**

if ( e.getSource() == empIDField ) {

**empIDNumber = Integer.parseInt( empIDField.getText() );**

if ( empIDNumber < 1 || empIDNumber > 20 ) {

**empIDField.setText( "Invalid employee ID" ); return;**

}

try {

**file.seek( ( empIDNumber - 1 ) \* EmpRecord.size() ); data.read( file );**

}

catch ( IOException ioe ) {

empIDField.setText( "Error reading file" );

}

if ( data.getEmpID() == 0 )

**empIDField.setText( empIDNumber + "does not exist" );**

}

else if ( e.getSource() == delete ) {

try {

**file.seek( ( empIDNumber - 1 ) \* EmpRecord.size() );**

**data.setEmpID( 0 ); data.setEmpName( "" );**

**data.setEmpDepartment( "" ); data.setPayrate( 0.0 ); data.write( file );**

}

catch ( IOException ioe ) {

empIDField.setText( "Error writing file" );

}

setVisible( false ); empIDField.setText( "" );

}

else if ( e.getSource() == cancel ) {

setVisible( false ); empIDField.setText( "" );

}

}

}

// Simple linked list in Ada

with Ada.Text\_IO; use Ada.Text\_IO;

procedure lsort1 is

type node;

type link is access node;

type node is

record

value: integer;

rest: character;

next: link;

end record;

package IntIO is new Ada.Text\_IO.Integer\_IO(integer); use IntIO;

int: integer;

l: link;

pt: array(1..10) of link;

ch: character;

begin

for i in 1..10 loop pt(i) := null; end loop;

loop

put("enter an integer key (1 thru 10), 99 to stop ");

get(int);

exit when int = 99;

put("enter th other infor, 1 char: ");

get(ch);

pt(int) := new node'(int,ch,pt(int));

end loop;

-- traverse list iteratively

for i in 1..10 loop

l := pt(i);

while l /= null loop

put(l.value); put(" "); put(l.rest);

new\_line;

l := l.next;

end loop;

end loop;

end lsort1;

// Simple linked list with procedures

with Ada.Text\_IO; use Ada.Text\_IO;

procedure klingon is

type cell;

type link is access cell;

type cell is

record

value: integer;

next: link;

end record;

package IntIO is new Ada.Text\_IO.Integer\_IO(integer): use IntIO;

int, sum: integer;

head,l: link;

procedure add\_to\_list(list: in out link; v: in integer) is

l:link;

begin

l := new cell;

l. value := v; l.next := list;

list := l; -- list := new cell'(v,list);

end;

-- main body of program to capture klingons

begin

head := null;

loop

put("enter an integer, 99 to stop "); get(int);

exit when int = 99;

add\_to\_list(head, int);

end loop;

-- traverse list iteratively

sum :=0;

l := head;

while l /= null loop

put(l.value); sum := sum + l.value;

new\_line; l := l.next;

end loop;

put("The sum of the elements is "); put(sum); new\_line;

end klingon;

-- in file Dostack.adb

with Ada.Text\_IO; use Ada.Text\_IO;**procedure dostack is** package MyInt\_IO is new Ada.Text\_IO.Integer\_IO(integer); use MyInt\_IO; m: integer;

max: constant := 100; -- Declare the stack

s: array(1..max) of integer;

top: integer range 0..max := 0;

**procedure push(x: integer) is**

begin

top := top + 1; s(top) := x;

**end push;**

**procedure pop(x: out integer) is**

begin

x := s(top); top := top - 1;

**end pop;**

begin

top := 0; --initialize top of stack to empty

for i in 1..4 loop

put("enter an integer"); get(m); **push(m**);

end loop;

for i in 1..4 loop

put("result of pop"); **pop(m);** put(m); new\_line;

end loop;

**end dostack;**

**C:\> Dostack**

enter an integer 3

enter an integer 5

enter an integer 36

enter an integer 41

result of pop 41

result of pop 36

result of pop 5

result of pop 3

-- in file stack.ads, body in stack.adb, main in usestack.adb**package stack is -- Example of “ADT” or Abstract Data Type!**

**procedure push(x:integer);**

**procedure pop(x: out integer);**

**end stack;**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

**package body stack is** -- in file stack.adb max: constant := 100; -- Data structures to create stack. s: array(1..max) of integer; top: integer range 0..max := 0;

**procedure push(x: integer) is**

begin

top := top + 1; s(top) := x;

**end push;**

**procedure pop(x: out integer) is**

begin

x := s(top); top := top - 1;

**end pop;**

begin

top := 0; --initialize top of stack to empty

**end stack;**

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

with Ada.Text\_IO; use Ada.Text\_IO; --in file usestack.adb**with stack; use stack;procedure usestack is** package IIO is new Ada.Text\_IO.Integer\_IO(integer); use IIO; m: integer;

begin

for i in 1..4 loop

put("enter an integer"); get(m);

**stack.push(m);**

end loop;

for i in 1..4 loop

put("result of pop");

**stack.pop(m);** put(m); new\_line;

end loop;

**end usestack;**

**Ada Generic Instantion**

Generics where introduced into Ada in 1985 to allow the compiler to tailor code that differs from other code only in the data type used. For example converting in integer stack to a float stack. Generics may be applied to procedures, functions, and packages. This feature makes a language data extensible.

procedure swap(x,y: in out float) is

t: float:

begin

t := x; x := y; y := t;

end;

------------------------------------------

**generic**

type item is private;

**procedure swap(x, y: in out item);** -- generic specification.

**procedure swap(x, y: in out item) is** -- generic body definition.

t: item;

begin

t := x; x := y; y := t;

**end;**

type month\_type is (jan, feb, mar, apr, may);

type date is record

month: month\_type; day: integer range 1..31; year: integer;

end record;

**procedure exchange is new swap( float );** -- instantiation.

**procedure exchange is new swap( date );** -- code written by compiler.

**procedure exchange is new swap( integer );**

**procedure charSwap is new swap( character );**

date1, date2: date; int1, int2: integer;

**exchange( date1, date2 );** -- Compile time binding based on parameter types.

**exchange( int1, int2 );**

**generic -- in file Gstack.ads**

max:integer; -- size of stack

type item is private; -- type to stack

**package gstack is**

**procedure push(x: in item);**

**procedure pop(x: out item);**

**end gstack;**

%%%%%%%%%%%%%%%%%%%%%%%%%%

**package body gstack is** -- in file Gstack.adb

s:array(1..max) of item;

top: integer range 0..max;

Sample run:

enter an integer: 10

enter an integer: 20

enter an integer: 30

enter an integer: 40

result of pop 40

result of pop 30

result of pop 20

result of pop 10

**procedure push(x: in item) is**

begin

top := top + 1; s(top) := x;

**end push;**

**procedure pop(x: out item ) is**

begin

x := s(top); top := top - 1;

**end pop;**

begin

top := 0; --initialize top of stack to empty

**end gstack;**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

with Ada.Text\_IO; use Ada.Text\_IO; -- in file Gusestac.adb

**with gstack;** -- generic stack defined in gstack10.ads /.adb**procedure genstack is** package IIO is new Ada.Text\_IO.Integer\_IO(integer); use IIO;

**package integer\_stack is new gstack(100,integer); use integer\_stack;** m: integer;begin for i in 1..4 loop put("enter an integer "); get(m); **push(m);** end loop; for i in 1..4 loop put("result of pop "); **pop(m);** put(m); new\_line;

end loop;

**end genstack;**

**-- Consider adding**: **package intStk is new gstack(10,integer); use intStk;-- Now push(m) will result in a compile time error as “Push” cannot be**

**--uniquely determined from the context. Use intStk.push(m) or**

**--integer\_stack.push(m), the full object oriented notation.**

Sample class definition of a sequentially allocated generic circular queue in Ada. The user must specify the data type. The size of the queue is defaulted to 24 if not specified. “Send” places an entry in the queue. “Receive” removes an entry from the queue. If the queue is full, “send” is ignored. If the queue is empty, garbage is returned for “receive.”

--in file SMailbox.ads. For use with UseSMail.adb **generic** type message is private; capacity: in natural := 24; -- Default queue size **package SMailbox is procedure send(msg: in message);**

**procedure receive(msg: out message);**

**end SMailbox;**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

**package body SMailbox is** -- in file Smailbox.adb subtype slotindex is integer range 0..(capacity - 1); head, tail: slotindex := 0;

mesnum : integer range 0..capacity := 0; -- number in buff

box: array(slotindex) of message; -- circular buffer

**procedure send(msg: in message) is**

begin

if mesnum < capacity then

box(head) := msg; head := (head+1) mod capacity;

mesnum := mesnum + 1;

end if;

**end send;**

**procedure receive(msg: out message) is**

begin

if mesnum > 0 then -- remove message if buff not empty

msg := box(tail); tail := (tail + 1) mod capacity;

mesnum := mesnum - 1;

end if;

**end receive;**

**end SMailbox;**

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

with Ada.Text\_IO; use Ada.Text\_IO; -- in file UseSMailbox.adb **with SMailbox;** procedure UseSMailbox is **package char\_mailbox is new SMailbox(character);**

**use char\_mailbox;**

m : character;

**package integerMailbox is new SMailbox(integer, 6);**

**use integerMailbox;**

n: integer;

package iio is new Ada.Text\_IO.Integer\_IO(integer); use iio;

begin

for i in character('a')..'z' loop put(i); send(i); end loop;

new\_line;

for i in character('a')..'z' loop receive(m); put(m); end loop;

new\_line;

-- char\_mailbox.send(i) and char\_mailbox.receive(m) may be used

-- if desired to emphasize use of the class object. The dot notation

-- must be used if the “use” clauses are not included.

for i in character('A')..'D' loop put(i); send(i); end loop;

new\_line;

for i in character('A')..'D' loop receive(m); put(m); end loop;

new\_line;

for i in 1..10 loop put(i,4); send(i); end loop;

new\_line;

for i in 1..10 loop receive(n); put(n,4); end loop;

new\_line;

end UseSMailbox;

c:\>UseSMail

abcdefghijklmnopqrstuvwxyz

abcdefghijklmnopqrstuvwx

ABCD

ABCD

1 2 3 4 5 6 7 8 9 10

1 2 3 4 5 6 18 18 18 18

with Ada.Text\_IO; use Ada.Text\_IO;

procedure box1 is

package IIO is new Ada.Text\_IO.Integer\_IO(integer);

use IIO;

type vector is array(integer range <>) of integer;

i: integer;

begin

loop

put("Enter array size ");

get(i);

**declare**

**a:vector(1..i); -- dynamically allocate storage in stack**

**begin**

**for k in a'range loop**

**put("enter the next integer ");**

**get(a(k)); new\_line;**

**end loop;**

**for k in a'range loop**

**put(a(k)); new\_line;**

**end loop;**

**end;**

put("back in main pgm with storage deallocated"); new\_line;

end loop;

end;

\*\* Storage allocation and reclaimation in the heap is the programmers responsibility. Storage allocation in the stack and especially storage reclaimation is done very efficiently by the runtime system!

This is an abstract example to show how to create generic arrays. The fact that it is an abstract example results in complexity not normally encountered.

%%%%%%%%%%%%%%%%%%%%

-- in file genericArray.ads The show how to create and export an array

-- of any type using subscripts of any enumeration type desire.

**generic**

**type subscript is (<>);**

**type myType is private;**

**package genericArray is**

**type userDefinedArray is array(subscript range <>) of myType;**

**end genericArray;**

%%%%%%%%%%%%%%%%%%%%%%%%

-- in file useGenericArray.adb

with ada.text\_io; use ada.text\_io;

**with genericArray;**

**procedure useGenericArray is**

type month\_name is (jan, feb, mar, apr, may, jun,

jul, aug, sep, oct, nov, dec);

type date is

record

day: integer range 1..31; month: month\_name; year: integer;

end record;

type family is (mother, father, child1, child2, child3, child4);

package month\_name\_io is new ada.text\_io.enumeration\_io(month\_name);

use month\_name\_io;

**package createShotArrayType is new genericArray(family, date);**

**use createShotArrayType;**

**vaccine: userDefinedArray(mother..child2);**

begin

vaccine(child2).month := jan; vaccine(child2).day := 22;

vaccine(child2).year := 1986;

put(jan); new\_line;

put(vaccine(child2).month); new\_line;

put("enter child1 month of birth ");

get(vaccine(child1).month);

put(vaccine(child1).month); new\_line;

end useGenericArray;

-- in file CompStk1.ads

with Ada.Text\_IO; use Ada.Text\_IO;

package CompStk1 is

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer);

use IntIO;

type MonthName is (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep,

Oct, Nov, Dec);

package MonthNameIO is new

Ada.Text\_IO.Enumeration\_IO(MonthName);

use MonthNameIO;

type Date is record

Month: MonthName;

Day: Integer range 1..31;

Year: Integer range 1700..2020;

end record;

procedure PrintDate(aDate: Date);

end CompStk1;

--in file CompStk1.adb

package body CompStk1 is

procedure PrintDate(aDate: Date) is

begin

put("mmm/dd/yyyy is "); put(aDate.Month);put("/");

put(aDate.Day,2); put("/"); put(aDate.Year,4);

end PrintDate;

end CompStk1;

-- in file UseCmpS1.adb

with CompStk1; use CompStk1;

with Ada.Unchecked\_Deallocation;

with Ada.Text\_IO; use Ada.Text\_IO;

procedure UseCmpS1 is

type Node; -- Avoid recursive definition.

type NodePtr is access Node;

type Node is record

HireDate: Date;

Next: NodePtr;

end record;

procedure Free is -- Provide procedure to return node to available storage list.

new Ada.Unchecked\_Deallocation( Node, NodePtr );

Head, Pt: NodePtr; --Pointers to stack set null initially.

begin

-- Insert first node (push).

Pt := new Node; -- Allocate node, “new” returns null for overflow.

Pt.HireDate.Month := Aug; -- Insert information.

Pt.HireDate.Day := 15; Pt.HireDate.Year := 1978;

Pt.Next := Head; Head := Pt; -- Insert in stack.

-- Insert second node (push).

Head := new Node'((Feb, 23, 2020), Head);

-- Insert third node.

Head := new Node'((Dec, 24, 1950), Head);

**Sample output:**

mmm/dd/yyyy is DEC/24/1950

mmm/dd/yyyy is FEB/23/2020

mmm/dd/yyyy is AUG/15/1978

while Head /= null loop

PrintDate(Head.HireDate); new\_line;

Pt := Head;

Head := Head.Next; -- Pop the stack.

Free( Pt ); -- Avoid hemorraging.

end loop;

End;

generic -- in file CompStkg.ads

type **MyType** is private;

package CompStkg is

procedure Push(X: MyType);

function Pop return MyType;

**private**

type Node; -- Avoid recursive definition.

type NodePtr is access Node;

type Node is record

MyData: **MyType**;

Next: NodePtr;

end record;

end CompStkg;

with **Ada.Unchecked\_Deallocation**; -- in file CompStkg.adb

package body CompStkg is

function free is new Ada.Unchecked\_Deallocation(Node, NodePtr);

Head, pt: NodePtr := null;

procedure Push(X: MyType) is

begin -- No check for overflow.

Head := new Node'(X, Head);

end Push;

function Pop return MyType is

X: MyType;

begin

X := Head.MyData; -- No check for underflow.

pt := Head;

Head := Head.Next;

free(pt); -- Note the hemorraging of memory if forgotten.

return X;

end Pop;

end CompStkg;

-- Example of programming by "Composition" (bottom-up)

-- as opposed to programming by "Classification"

-- (top-down) better know as the use of inheritance.

with Ada.Text\_IO; use Ada.Text\_IO;

with CompStk1; use CompStk1;

with CompStkg;

procedure UCmpStkg is -- in file UCmpStkg

package CharStack is new CompStkg(character);

use CharStack;

package DateStack is new CompStkg(CompStk1.Date);

use DateStack;

Char: Character;

ADate: Date;

begin

Push('A'); Push('B'); Push('C');

put(pop); put(Pop); put(Pop); new\_line(2);

Push((Jan, 15, 1992)); Push((Mar, 24, 1994));

Push((Jun, 12, 1999));

ADate := Pop; PrintDate(ADate); new\_line;

Adate := Pop; PrintDate(Adate); new\_line;

Adate := Pop; PrintDate(ADate); new\_line;

end;

**Sample Output:**

CBA

mmm/dd/yyyy is JUN/12/1999

mmm/dd/yyyy is MAR/24/1994

mmm/dd/yyyy is JAN/15/1992

--in file taged1.adb

with Ada.Integer\_Text\_IO; use Ada.Integer\_Text\_IO;

**procedure taged1 is**

**type Rectangle is tagged**

**record**

**length: Integer;**

**width: Integer;**

**end record;**

**function Size(r: in Rectangle) return Integer is**

begin return r.length \* r.width; **end Size;**

-- create a cube by inheriting from Rectangle.

***type Cube is new Rectangle with***

***record height: Integer; end record;***

-- Cube inherits fields length, width, and function Size.

-- This size may be redefined as:

**function Size(c: Cube) return Integer is**

begin

return **Size(** **Rectangle(** C **)** **)** \* C.height;

**end Size;**

-- Note the type conversion "Rectangle(c)" so that the inherited

-- function Size for Rectangle (overload) can be used.

**rect1: Rectangle := (6,10);**

**cube1: Cube := (length => 6, width => 10, height => 20);**

begin

**put( Size( rect1 ) ); put( Size( cube1 ) );**

**rect1 := Rectangle( cube1 ); cube1 := ( rect1 with 20 );**

end taged1;

-- User-written subprograms are classified as primitive operations

-- if they are declared in the same package specification as the

-- type and have the type as a parameter or result. Derived types

-- inherit all primitive operations that belong to the parent type.

-- Programing by “Classification” (top-down) as opposed to

-- composition by “Composition” (bottom-up).

package AbstStck is

type AbstractStack is limited private;

type AbstractStackElement is tagged private;

type AbstractStackElementPtr is

access all AbstractStackElement'Class;

procedure Push(Stack: access AbstractStack; Y: in AbstractStackElementPtr);

function Pop(Stack: access AbstractStack) return AbstractStackElementPtr;

function StackSize(Stack: AbstractStack) return integer;

private

type AbstractStackElement is tagged

record

Next: AbstractStackElementPtr;

end record;

type AbstractStack is limited

record

Count: integer := 0;

Top: AbstractStackElementPtr := null;

end record;

end AbstStck;

package body AbstStck is

procedure Push(Stack: access AbstractStack; Y: in AbstractStackElementPtr) is

Pt: AbstractStackElementPtr;

begin

Y.Next := Stack.Top; Stack.Top := Y; Stack.Count := Stack.Count + 1;

end Push;

function Pop(Stack: access AbstractStack) return AbstractStackElementPtr is

Pt: AbstractStackElementPtr;

begin

if Stack.Top = null then -- Check for underflow.

return null;

end if;

Stack.Count := Stack.Count - 1;

Pt := Stack.Top; Stack.Top := Stack.Top.Next; -- Pop stack, note hemmoraging.

return Pt;

end Pop;

function StackSize(Stack: AbstractStack) return integer is

begin return Stack.Count; end StackSize;

end AbstStck;with AbstStck;

package MakeCar is

type String4 is new String(1..4);

type Car is new AbstStck.AbstractStackElement with record

NumDoors: integer;

Manufacturer: String4 := "GMC ";

end record;

procedure AssignNumDoors(aCar: in out Car; N: in integer);

procedure AssignManufacturer(aCar: in out Car; Manu: in String4);

procedure PrintNumDoors(aCar: in Car);

procedure PrintManufacturer(aCar: in Car);

procedure IdentifyVehicle(aCar: in Car);

end MakeCar;

with Ada.Text\_IO; use Ada.Text\_io;

with AbstStck;

package body MakeCar is

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer); use IntIO;

procedure AssignNumDoors(aCar: in out Car; N: in integer) is

begin aCar.NumDoors := N; end AssignNumDoors;

procedure AssignManufacturer(aCar: in out Car; Manu: in String4) is

begin aCar.Manufacturer := Manu; end AssignManufacturer;

procedure PrintNumDoors(aCar: in Car) is

begin put("Num doors = "); put(aCar.NumDoors); new\_line; end PrintNumDoors;

procedure PrintString4(PrtStr: String4) is

begin for I in 1.. 4 loop

put(PrtStr(I));

end loop; end PrintString4;

procedure PrintManufacturer(aCar: in Car) is

begin put("Manufacturer is "); PrintString4(aCar.Manufacturer); new\_line; end;

procedure IdentifyVehicle(aCar: in Car) is

begin

put("Car with "); put(aCar.NumDoors, 4); put(" doors");

put(" made by "); PrintString4(aCar.Manufacturer); new\_line;

end IdentifyVehicle;

end MakeCar;

with Ada.Text\_IO; use Ada.Text\_io;

with AbstStck; use AbstStck;

with MakeCar; use MakeCar;

procedure UAbstStc is

type Stack\_Ptr is access AbstractStack;

CarStack: Stack\_Ptr := new AbstractStack;

StackPoint: Stack\_Ptr;

NewCar, CarPt: AbstractStackElementPtr;

begin --Create 1st car.

NewCar := new Car'(AbstractStackElement with 4, "Ford");

push(CarStack, NewCar);

NewCar := new Car; -- Create 2nd car.

AssignNumDoors(Car'Class(NewCar.All), 2);

AssignManufacturer(Car'Class(NewCar.all), "Chev");

push(CarStack, NewCar);

NewCar := new Car; -- Create 3rd car.

AssignNumDoors(Car'Class(NewCar.All), 2);

-- Default manufacturer to "GMC ".

push(CarStack, NewCar);

for I in 1..StackSize(CarStack.all) loop

CarPt := pop(CarStack);

PrintManufacturer(Car'Class(CarPt.All));

PrintNumDoors(Car'Class(CarPt.All));

new\_line;

**Sample Output:**

Manufacturer is GMC

Num doors = 2

Manufacturer is Chev

Num doors = 2

Manufacturer is Ford

Num doors = 4

end loop;

end UAbstStc;

with AbstStck;

package MakePlane is

type String8 is new String(1..8);

type Plane is new AbstStck.AbstractStackElement with record

NumDoors: integer;

NumEngines: integer;

Manufacturer: String8 := "Boeing ";

end record;

procedure AssignNumDoors(aPlane: in out Plane; N: in integer);

procedure AssignManufacturer(aPlane: in out Plane; Manu: in String8);

procedure AssignNumEngines(aPlane: in out Plane; NE: in integer);

procedure PrintPlane(aPlane: in Plane);

procedure IdentifyVehicle(aPlane: in Plane);

end MakePlane;

with Ada.Text\_IO; use Ada.Text\_io; with AbstStck;

package body MakePlane is

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer); use IntIO;

procedure AssignNumDoors(aPlane: in out Plane; N: in integer) is

begin aPlane.NumDoors := N; end AssignNumDoors;

procedure AssignManufacturer(aPlane: in out Plane; Manu: in String8) is

begin aPlane.Manufacturer := Manu; end AssignManufacturer;

procedure AssignNumEngines(aPlane: in out Plane; NE: in integer) is

begin aPlane.NumEngines := NE; end AssignNumEngines;

procedure PrintString8(PrtStr: String8) is

begin for I in 1..8 loop put(PrtStr(I)); end loop; end PrintString8;

procedure PrintPlane(aPlane: in Plane) is

begin

put("Num doors for plane = "); put(aPlane.NumDoors, 4); new\_line;

put("Number engines = "); put(aPlane.NumEngines); new\_line;

put("Manufacturer = "); PrintString8(aPlane.Manufacturer); new\_line;

end PrintPlane;

procedure IdentifyVehicle(aPlane: in Plane) is

begin

put("Plane with "); put(aPlane.NumDoors, 4); put(" doors, ");

put(aPlane.NumEngines, 4); put(" engines, made by ");

PrintString8(aPlane.Manufacturer); new\_line;

end IdentifyVehicle;

end MakePlane;

with Ada.Text\_IO; use Ada.Text\_io;

with AbstStck; use AbstStck;

with MakeCar, MakePlane; use MakeCar, MakePlane;

procedure UAbstSt2 is

type Stack\_Ptr is access AbstractStack;

VehicleStack: Stack\_Ptr := new AbstractStack;

StackPoint: Stack\_Ptr;

NewCar, CarPt, NewPlane, PlanePt, VehiclePt:

AbstractStackElementPtr;

begin

NewCar := new Car'(AbstractStackElement with 4, "Ford");

push(VehicleStack, NewCar); -- 1st car.

NewPlane := new Plane'(AbstractStackElement with 2, 2, "Northrup");

push(VehicleStack, NewPlane); --1st plane.

for I in 1..StackSize(VehicleStack.all) loop

VehiclePt := pop(VehicleStack);

if VehiclePt.all in Car then *-- \*\* Identify class of object at run time.*

IdentifyVehicle(Car'Class(VehiclePt.all));

elsif VehiclePt.all in Plane then

IdentifyVehicle(Plane'Class(VehiclePt.all));

end if;

new\_line;

end loop;

end UAbstSt2;

**Sample Output:**

Plane with 2 doors, 2 engines, made by Northrup

Car with 4 doors made by Ford

**\*\*\*\* Heterogeneous versus Homogeneous!****//stack0.cpp THIS CLASS CREATES AN INTEGER STACK**

#include<iostream.h>

#define boolean int

**class stack0 {**

private:

enum {EMPTY = -1};

int \*s;

int max\_len;

int top;

**public:**

**stack0( ) :max\_len(20)**

**{** s = new int[max\_len]; top = EMPTY; **}**

**stack0( int size )**

**{** s = new int[size]; top = EMPTY; max\_len = size;**}**

**~stack0( ){** delete [ ] s; **}**

**void reset( ){** top = EMPTY;}

**void push( int x){** if(top < (max\_len - 1)) s[++top] = x; }

**int pop()** { if(top != EMPTY) return(s[top--]);else return(0);}

**int top\_of( );**

**boolean empty()** { return boolean( top == EMPTY); }

**boolean full()** { return boolean( top == max\_len - 1); }

**};**

**stack0::top\_of( )** { return (s[top]);}

**void main(void)**{

int j, num\_int;

**stack0** **int\_stack1(10);**

cout << "How many integers do you wish to stack? "; cin >> num\_int;

for( int i = 1; i <= num\_int; ++i){

cout << "enter data: "; cin >> j; **int\_stack1.push(j);**

}

cout << "\n\nPop the stack\n";

while( **!int\_stack1.empty()** ){ j = **int\_stack1.pop()**;

cout << "value = " << j << ".\n";

}

**}**

//stack1.cpp C++ introduced “templates” to allow generics.

#include<iostream.h>

#define boolean int

**template <class TYPE>**

**class stack {**

**private:**

enum {EMPTY = -1};

**TYPE**\* s;

int max\_len;

int top;

**public:**

**stack() :max\_len(20)**

**{** s = new TYPE[20]; top = EMPTY; **}**

**stack(int size) :max\_len(size)**

**{** s = new TYPE[size]; top = EMPTY;}

**~stack(){ delete [ ] s; }**

**void reset( )** { top = EMPTY;}

**void push(** **TYPE** x **)** { if(top < (max\_len - 1)) s[++top] = x; }

**TYPE pop( )** { if(top != EMPTY) return(s[top--]); else return (0);}

**TYPE top\_of();**

**boolean empty( )** { return boolean( top == EMPTY); }

boolean full(){ return boolean( top == max\_len - 1); }

**};**

**template<class TYPE> TYPE stack<TYPE> :: top\_of( )** { return (s[top]);}

**void main(void)**

**{** int j, num\_int;

**stack<int>** **int\_stack1(10);**

cout << "How many integers do you wish to stack? ";

cin >> num\_int;

for( int i = 1; i <= num\_int; ++i){

cout << "enter data: "; cin >> j;

**int\_stack1.push( j );**

}

cout << "\n\nPop the stack\n";

for( i = 1; i <= num\_int; ++i){

j = **int\_stack1.pop( )**; cout << "value = " << j << ".\n";

}

**}**

//stack2.cpp

#include<stdio.h>

#include<iostream.h>

#define boolean int

template <class TYPE>

class stack {

private:

enum {EMPTY = -1};

TYPE\* s;

int max\_len;

int top;

public:

stack() :max\_len(20)

{ s = new TYPE[20]; top = EMPTY;}

stack(int size) :max\_len(size)

{ s = new TYPE[size]; top = EMPTY;}

~stack(){ delete [] s;}

void reset(){ top = EMPTY;}

void push( TYPE x){ if(top < (max\_len - 1)) s[++top] = x; }

TYPE pop(){ if(top != EMPTY) return(s[top--]); else return(0);}

TYPE top\_of();

boolean empty(){ return boolean( top == EMPTY); }

boolean full(){ return boolean( top == max\_len - 1); }

};

template<class TYPE> TYPE stack<TYPE>::top\_of(){ return (s[top]);}

void main(void)

{ int j, num\_int;

char string1[20];

stack<int> int\_stack1(10);

stack<int> int\_stack2;

stack<char\*> string\_stack(5);

char\* char\_array[3];

char\_array[0] = "sam"; char\_array[1] = "mary"; char\_array[2] = "bob";

cout << "How many integers do you wish to stack in stack1? ";

cin >> num\_int;

for( int i = 1; i <= num\_int; ++i){

cout << "enter data: "; cin >> j;

int\_stack1.push(j);

}

cout << "\n\nPop the stack\n";

for( i = 1; i <= num\_int; ++i){

j = int\_stack1.pop(); cout << "value = " << j << ".\n";

}

cout << "How many integers do you wish to stack in stack2? ";

cin >> num\_int;

for( i = 1; i <= num\_int; ++i){

cout << "enter data: "; cin >> j;

int\_stack2.push(j);

}

cout << "\n\nPop the stack\n";

for( i = 1; i <= num\_int; ++i){

j = int\_stack2.pop(); cout << "value = " << j << ".\n";

}

cout << "\n\nStack three strings. \n";

num\_int = 3;

for( i = 0; i < num\_int; ++i){

cout << "push " << char\_array[i] << ".\n";

string\_stack.push(char\_array[i]);

}

cout << "\n\nPop the stack\n";

for( i = 1; i <= num\_int; ++i){

cout << "value = "; printf("%s\n", string\_stack.pop());

}

}

//stack3.cpp

#include<iostream.h>

#define boolean int

template <class TYPE>

class stack {

private:

enum {EMPTY = -1};

TYPE\* s;

int max\_len;

int top;

public:

stack( ) :max\_len(20)

{ s = new TYPE[20]; top = EMPTY;}

stack(int size) :max\_len(size)

{ s = new TYPE[size]; top = EMPTY;}

~stack(){ delete [] s;}

void reset( ){ top = EMPTY;}

void push( TYPE x){ if(top < (max\_len - 1)) s[++top] = x; }

TYPE pop( ){ if(top != EMPTY) return(s[top--]); else return(0);}

TYPE top\_of( );

boolean empty( ){ return boolean( top == EMPTY); }

boolean full( ){ return boolean( top == max\_len - 1); }

};

template<class TYPE> TYPE stack<TYPE>::top\_of(){ return (s[top]);}

#include<math.h>

class complex {

private:

double real, imag;

public:

complex(double r = 0.0, double i = 0.0)

{ real = r; imag = i;}

void assign(double r, double i){ real = r; imag = i;}

void print( ){ cout << real << " + " << imag << "i\n";}

complex operator=( const complex& n)

{complex temp; real=n.real;imag=n.imag; return temp;}

complex operator+( const complex& n)

{ complex temp(real + n.real, imag + n.imag); return temp;}

complex operator-(const complex& n)

{ complex temp(real - n.real, imag - n.imag); return temp;}

};

void main(void)

{ int j, num\_int;

stack<int> int\_stack1(10);

stack<complex> complex\_stack;

complex c1(2.0, 4.6), c2(3.5, 6.4), c3(-7.8, 9.2);

cout << "push c1 = 2.0 + 4.6i \n"; complex\_stack.push(c1);

cout << "push c2 = 3.5 + 6.4i \n"; complex\_stack.push(c2);

cout << "push c3 = -7.8 + 9.2i \n"; complex\_stack.push(c3);

cout << "\n\n Pop the complex numbers and print.\n";

complex\_stack.pop().print(); cout << "\n";

complex\_stack.pop().print(); cout << "\n";

complex\_stack.pop().print(); cout << "\n";

cout << "How many integers do you wish to stack? ";

cin >> num\_int;

for( int i = 1; i <= num\_int; ++i){

cout << "enter data: "; cin >> j;

int\_stack1.push(j);

}

cout << "\n\nPop the stack\n";

for( i = 1; i <= num\_int; ++i){

j = int\_stack1.pop(); cout << "value = " << j << ".\n";

}

}

//stack4.cpp generic stack

#include<iostream.h>

#define boolean int

template <class TYPE>

class stack {

private:

enum {EMPTY = -1};

TYPE\* s;

int max\_len;

int top;

public:

stack() :max\_len(20)

{ s = new TYPE[20]; top = EMPTY;}

stack(int size) :max\_len(size)

{ s = new TYPE[size]; top = EMPTY;}

~stack(){ delete [ ] s;}

void reset(){ top = EMPTY;}

void push( TYPE x){ if(top < (max\_len - 1)) s[++top] = x; }

TYPE pop(){ if(top != EMPTY) return(s[top--]); else return (0);}

TYPE top\_of();

boolean empty(){ return boolean( top == EMPTY); }

boolean full(){ return boolean( top == max\_len - 1); }

};

template<class TYPE> TYPE stack<TYPE>::top\_of(){ return (s[top]);}

void main(void)

{ int j, num\_int;

cout << "How much space should be allocated to the stack? "; cin >> j;

stack<int> int\_stack1(j); // Dynamic allocation of stack space.

cout << "How many integers do you wish to stack? ";

cin >> num\_int;

for( int i = 1; i <= num\_int; ++i){

cout << "enter data: "; cin >> j; int\_stack1.push(j);

}

cout << "\n\nPop the stack\n";

for( i = 1; i <= num\_int; ++i){

j = int\_stack1.pop(); cout << "value = " << j << ".\n";

}

}

-- In file GIOEX.ads**. Creates a rectangle with user defined data type for length and**

**-- width.**  **Note that it is frequently desirable to pass methods including I/O routines**

**-- for programmer defined data types.**

-- The following demonstrates how to pass I/O procedures to a generic package.

-- The rectangle may also be used for **inheritance** if desired.

**generic**

**type MyType is private;**

**with function "\*"(X,Y: MyType) return MyType;**

**with procedure Put(X: MyType);**

**package GIOEX is**

type Rectangle is **tagged**

record

Length: MyType;

Width: MyType;

end record;

function Size(r: in Rectangle) return MyType; -- intrinsic functions

function RectLength(r: in Rectangle) return MyType;

**end GIOEX;**

--in file GIOEX.adb

with Ada.Text\_IO; use Ada.Text\_IO; -- Access restricted to body.

**package body GIOEX is**

**function Size(r: in Rectangle) return MyType is**

begin

put("The Size of the Rectangle with length "); **put(r.Length);**

put(" and width "); **put(r.Width);** put(" is ");

**put( r.Length \* r.Width);** put("!"); new\_line(2);

return **r.Length \* r.Width**;

**end Size;**

**function RectLength(r: in Rectangle) return MyType is**

begin

put("The length is "); **put(r.Length);**

new\_line;

return r.Length;

end;

**end GIOEX;**

-- Sample program to show how to pass a programmer defined data type

-- and I/O methods to a generic.

-- in file UGIOEX.adb

**with GIOEX;**

with Ada.Text\_IO; -- Use Ada.Text\_IO;

**procedure UGIOEX is**

package MyFloatIO is new Ada.Text\_IO.Float\_IO(Float);

use MyFloatIO;

--The generic put statement format in Ada.Text\_IO.Float\_IO.

-- procedure Put( item: float;

-- fore: Ada.Text\_IO.field := 0; -- “0” means use minimum space.

-- aft: Ada.Text\_IO.Field := 0;

-- exp: Ada.Text\_IO.Field := 0

-- );

-- Supply an overload for the generic written by the compiler in MyFloat\_IO.

**procedure MyPut(X: Float) is**

**begin MyFloatIO.Put(X, 0, 0, 0); end;**

-- Note that “\*” is defined for the intrinsic type Float.

**package MyGIOEX is new GIOEX( Float, MyPut, "\*");**

**use MyGIOEX;**

Rect1: Rectangle := (5.0, 6.0); //Create class object using constructor!

Len: Float;

begin

Len := Size(Rect1); Len := RectLength(Rect1);

end UGIOEX;

c:\>UGIOEX

The Size of the Rectangle with length 5.0 and width 6.0 is

30.0! -- 3.00000E+01

The length is 5.0 -- 5.00000E+00

-- in file usegioex2.adb. Use of Venus units for measurement.

-- These operations are required for our human mission to Venus.

with Ada.Text\_IO; use Ada.Text\_io;

**with GIOEX;**

procedure UseGioex2 is

package MyIntIO is new Ada.Text\_IO.Integer\_IO(Integer);

package MyFloatIO is new Ada.Text\_IO.Float\_IO(Float);

**type VenusMeasure is record**

**F1: Integer;**

**F2: Float;**

**end record;**

-- Define I/O for VenusMeasurement.

**procedure Put(v: VenusMeasure) is**

**begin**

**MyIntIO.put(v.F1);**

**put(" ");**

**MyFloatIO.put(v.F2);**

**new\_line;**

**end;**

-- Define mutliplication for VenusMeasurement.

**function "\*"(p1: VenusMeasure; p2: VenusMeasure)**

**return VenusMeasure is**

**temp: VenusMeasure;**

**begin**

**temp.F1 := p1.F1 \* p2.F1;**

**temp.F2 := p1.F2 \* p2.F2;**

**return temp;**

**end;**

**package MyVenusRectangle**

**is new GIOEX(VenusMeasure, Put, "\*");**

**use MyVenusRectangle;**

width: VenusMeasure := ( 5, 5.5);

height: VenusMeasure := (3, 2.4);

Rect1: Rectangle := (width, height); -- Creates a rectangle using

-- Venus measurements.

Ans: VenusMeasure;

begin

Ans := Size( Rect1 );

end UseGIOEx2;

\*\*\* The generic for Enumeration\_IO follows: \*\*\*

ada io from arm

with Ada.Text\_IO; use Ada.Text\_IO;

**procedure alphtre1 is**

**type Node;**

**type Tree is access Node;**

**type Node is**

**record**

**Value: Character;**

**Left, Right: Tree;**

**end record;**

Ch: character;

**Root: tree;**

**procedure Insert(t: in out Tree; v: Character) is --** build the tree

begin

if t = null then

t := new Node; t.Value := v;

t.Left := null; -- actually set to null automatically by Ada

t.Right := null;

else

if v < t.Value then

Insert(t.Left,v);

else

Insert(t.Right,v);

end if;

end if;

**end Insert;**

**procedure Inorder(t:Tree) is -- inorder tree traversal**

**begin**

**if t /= null then**

**Inorder(t.Left); put("tree sort "); // 1 Traverse Left**

**put(t.Value); new\_line; // 2 Visit**

**inorder(t.Right); // 3 Traverse Right**

**end if;**

**end Inorder;**

begin -- read info and place in sorted order recursively

root := null; -- All pointers set to null initially automatically in ada

loop

put("Enter a character, 'Z' to exit: "); get(Ch); exit when Ch = 'Z';

Insert(Root, Ch);

end loop;

Inorder(root); -- print in sorted order

end;

**with Ada.Text\_IO; use Ada.Text\_IO;**

**procedure alphtre2 is -- Builds tree recursively.**

**type Node; -- Traverses iteratively.**

**type Tree is access Node;**

**type Node is**

**record**

**Value: Character; Left, Right: Tree;**

**end record;**

**Ch: character;**

**Root: tree;**

**procedure Insert(t: in out Tree; v: Character) is -- build the tree**

**begin**

**if t = null then**

**t := new Node; t.Value := v;**

**t.Left := null; t.Right := null;** -- Actually the default in Ada is null when allocated.

**else**

**if v < t.Value then**

**Insert(t.Left,v);**

**else**

**Insert(t.Right,v);**

**end if;**

**end if;**

**end Insert;**

**procedure Inorder(t:Tree) is** -- Iterative inorder tree traversal**.**

**Stack: array(1..10) of Tree;** -- Balance tree requires log base 2

**Knt: integer; --** of N space, N is number nodes.

**Pt: Tree;**

**begin**

**Knt := 0; Pt := t;-- Set stack empty, Pt to the root of the tree.**

**loop**

**if Pt /= null then**

**Knt := Knt + 1; Stack(Knt) := Pt; Pt := Pt.Left;**

**else**

**exit when Knt = 0; -- Traveresed whole tree.**

**Pt := Stack(Knt); Knt := Knt - 1;**

**put("tree sort "); put(Pt.Value); new\_line; Pt := Pt.Right;**

**end if;**

**end loop;**

**end Inorder;**

**begin** -- read info and place in sorted order recursively

**root := null;** -- All pointers set to null initially automatically in ada

**loop**

**put("Enter a character, 'Z' to exit: "); get(Ch); exit when Ch = 'Z';**

**Insert(Root, Ch);**

**end loop;**

**Inorder(root);** -- print in sorted order

**end;**

// This program creates a binary search tree, inserts data, then traverses in inorder.

// In file Btree.cpp

#include <iostream.h>

#define Null 0

**struct Node**

**{**

**char aChar;**

**struct Node \*Left;**

**struct Node \*Right;**

**};**

struct Node \*NewNode(const char Char)

{

struct Node \*Point;

Point = new Node;

Point->aChar = Char;

Point->Left = Null;

Point->Right = Null;

return Point;

}

typedef struct Node \*NodePtr;

void Insert(const char NewChar, NodePtr &Pt) //note CBR required.

{ cout << NewChar << " in insert\n";

if (Pt == Null)

{ cout << "insert first node " << NewChar << "\n";

Pt = NewNode(NewChar);

return;

}

if(NewChar < Pt->aChar)

if (Pt->Left == 0)

Pt->Left = NewNode(NewChar);

else

Insert(NewChar, Pt->Left);

else if(NewChar > Pt->aChar)

if(Pt->Right == 0)

Pt->Right = NewNode(NewChar);

else

Insert(NewChar, Pt->Right);

else

cerr << "\n#### Insert: " << NewChar << " is already in tree";

}

void InOrder(NodePtr Pt) // Note CBV required

{

if(Pt != Null)

{

if(Pt->Left != 0) InOrder(Pt->Left);

cout << "In order traversal: " << Pt->aChar <<"\n";

if(Pt->Right != 0) InOrder(Pt->Right);

}

}

void main() l

{ char Dummy;

struct Node \*Root = NULL;

cout << "hello\n";

Insert('f', Root);

Insert('e', Root);

Insert('b', Root);

Insert('d', Root);

Insert('a', Root);

Insert('c', Root);

Insert('g', Root);

InOrder(Root);

cin >> Dummy;

}

#include <iostream.h> // Builds and traverses a binary search tree.

#define Null 0 // in file BtreeCla.cpp

**class BinarySearchTree {**

private:

struct Node

{

char aChar;

struct Node \*Left;

struct Node \*Right;

};

typedef struct Node \*NodePtr;

struct Node \*Root;

struct Node \*NewNode(const char Char)

{

struct Node \*Point;

Point = new Node;

Point->aChar = Char;

Point->Left = Null;

Point->Right = Null;

return Point;

}

void InsertNode(const char NewChar, NodePtr &Pt) // CBR required.

{ cout << NewChar << " in insert\n";

if (Pt == Null)

{ cout << "insert first node " << NewChar << "\n";

Pt = NewNode(NewChar); return;

}

if(NewChar < Pt->aChar)

if (Pt->Left == 0)

Pt->Left = NewNode(NewChar);

else

InsertNode(NewChar, Pt->Left);

else if(NewChar > Pt->aChar)

if(Pt->Right == 0)

Pt->Right = NewNode(NewChar);

else

InsertNode(NewChar, Pt->Right);

else

cerr << "\n#### Insert: " << NewChar << " is already in tree";

}

void PrtInOrder(NodePtr Pt) // CBV required

{

if(Pt != Null)

{

if(Pt->Left != 0) PrtInOrder(Pt->Left);

cout << "In order traversal: " << Pt->aChar <<"\n";

if(Pt->Right != 0) PrtInOrder(Pt->Right);

}

}

public:

BinarySearchTree() { Root = Null;}

void Insert(const char NewChar){ InsertNode(NewChar, Root);}

void InOrder(){PrtInOrder(Root);}

}; // End class definition

// #include BinarySearchTree;

void main()

{

char Dummy;

BinarySearchTree BSTree;

BSTree.Insert('f'); BSTree.Insert('e');;

BSTree.Insert('b'); BSTree.Insert('d');

BSTree.Insert('a'); BSTree.Insert('c');

BSTree.Insert('g'); BSTree.Insert('z');

BSTree.InOrder();

cin >> Dummy;

}

#include <iostream.h> // in file BSTCTmp.cpp

#define Null 0

template <class DataType>

class BinarySearchTree {

private:

struct Node

{

DataType MyData;

struct Node \*Left;

struct Node \*Right;

};

typedef struct Node \*NodePtr;

struct Node \*Root;

struct Node \*NewNode(const DataType MyDataIn)

{

struct Node \*Point;

Point = new Node;

Point->MyData = MyDataIn;

Point->Left = Null;

Point->Right = Null;

return Point;

}

public:

BinarySearchTree() { Root = Null;}

void Insert(const DataType MyData){ InsertNode(MyData, Root);}

void InsertNode(const DataType MyData, NodePtr &Pt)

{ cout << MyData << " in insert\n";

if (Pt == Null)

{ cout << "insert first node " << MyData << "\n";

Pt = NewNode(MyData); return;

}

if(MyData < Pt->MyData)

if (Pt->Left == 0)

Pt->Left = NewNode(MyData);

else

InsertNode(MyData, Pt->Left);

else if(MyData > Pt->MyData)

if(Pt->Right == 0)

Pt->Right = NewNode(MyData);

else

InsertNode(MyData, Pt->Right);

else

cerr << "\n#### Insert: " << MyData << " is already in tree";

}

void InOrder(){PrtInOrder(Root);}

void PrtInOrder(NodePtr Pt)

{

if(Pt != Null)

{

if(Pt->Left != 0) PrtInOrder(Pt->Left);

cout << "In order traversal: " << Pt->MyData <<"\n";

if(Pt->Right != 0) PrtInOrder(Pt->Right);

}

}

}; // End class definition

##############################################

// #include BinarySearchTree;

void main()

{

char Dummy;

BinarySearchTree<char> BSTree;

BSTree.Insert('f'); BSTree.Insert('e');;

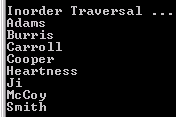
BSTree.Insert('b'); BSTree.Insert('d');

BSTree.Insert('a'); BSTree.Insert('c');

BSTree.Insert('g'); BSTree.Insert('z');

cout << "\n\n";

BSTree.InOrder();



cout << "\n\n";

BinarySearchTree<int> BSTint;

BSTint.Insert(13); BSTint.Insert(10);;

BSTint.Insert(22); BSTint.Insert(6);

BSTint.Insert(3); BSTint.Insert(12);

cout << "\n\n";

BSTint.InOrder();

cin >> Dummy;

}

File Organization

And

Access Methods

Organization:

1. Sequential
2. Indexed
3. Relative

Access Methods:

1. Sequential
2. Random
3. Dynamic

--file: RelativeFiles.doc

Random Access Methods

Records are stored and retrieved on the basis of a predictable relationship between the key of the desired record and the location where the record is stored.

Methods

1. Direct: The user supplies either the direct address on the storage device of the desired record (e.g., on a disk the cylinder, track, and sector) or a virtual address that can be transformed easily to the actual address either by the system software or device hardware (e.g., a relative record number for conversion).
2. Dictionary Look-up.
3. Hashing or Calculation.

Relative Files

Traditional – Fixed Length Records

All space for file is typically allocated as a contiguous unit at the time the file is created.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  | Betty | Adam |  | Sue |  | Tom | ••• |
| Record  Number | 0 | 1 | 2 | 3 | 4 | 5 |  |
| Device  Address | 0 | 50 | 100 | 150 | 200 | 250 |  |

Loc[RECJ ] = Base + Offset

Offset = J \* number of characters in record

**Example:** Find relative record 3 assuming 50 characters per record starting at a base address of zero on the storage device.

**LOC[ REC3 ] = 0 + 3 \* 50 = 150**.

&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

**Relative files may be based on a table lookup.** This is particularly attractive for:

1. variable length records,
2. in situations where dynamic storage allocation is desirable.

Storage Device Address

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Index |  |  |  |  |
|  | 0 | 336 |  | Adam | 105 |  |
|  | 1 | 105 |  |  |  |  |
|  | 2 |  |  |  |  |  |
|  | 3 | 768 |  | Betty | 336 |  |
|  | 4 |  |  |  |  |  |
|  | 5 |  |  |  |  |  |
|  | ••• |  |  | Sue | 768 |  |
|  |  |  |  |  |  |  |

-- Demonstrate sequential I/O of records in Ada.

with Ada.Text\_IO;

**with sequential\_io;**

procedure seqio is

type rec is record

i:integer;

a:string(1..5);

end record;

**package hope is new sequential\_io(rec);**

**use hope;**

**f1: file\_type;**

rec1: rec;

begin

**hope.create(f1,out\_file,"joe");**

rec1.a := "joeis";

rec1.i := 99;

**hope.write(f1,rec1);**

rec1.a := "samis";

rec1.i := 23;

**hope.write(f1,rec1);**

**hope.close(f1);**

**hope.open(f1,in\_file,"joe");**

while not **end\_of\_file(f1)** loop

**hope.read(f1,rec1);**

**-- process record**

end loop;

Ada.Text\_IO.put("hi there");

**hope.close(f1);**

end;

**with direct\_io;** //in file mkdirect.ada & directio.doc

with text\_io;

procedure mkdirect is

**type rec is record**

**i:integer;**

**a:string(1..5);**

**end record;**

package iio is new text\_io.integer\_io(integer);

use iio;

**package io\_direct is new direct\_io(rec);**

**use io\_direct;**

**pt: positive\_count;**

**f1:file\_type;**

**rec1: rec;**

j: integer := 0;

begin

**create(f1,inout\_file,"joedir");**

rec1.a := "abcde";

**reset(f1);**

-- COUNT and POSITIVE\_COUNT are defined in package direct\_io;

for pt in positive\_count range 1..10 loop

j := j + 1;

rec1.i := j;

**write(f1, rec1, pt);**

end loop;

**close(f1);**

**open(f1,inout\_file,"joedir");**

**reset(f1);**

for pt in positive\_count range 1..10 loop

**read(f1, rec1, pt);**

text\_io.put(rec1.a); iio.put(rec1.i); text\_io.new\_line;

end loop;

**close(f1);**

end;

with Ada.Text\_IO; use Ada.Text\_IO;

with Ada.Unchecked\_Deallocation;

procedure LinkQ is

type JobType is (Programmer, Manager, Accountant, Analysist,

Sales, Manufacturing, Inventory, SoftwareEngineer);

package JobTypeIO is new Ada.Text\_IO.Enumeration\_IO(JobType); use JobTypeIO;

type EmpName is (David, Kevin, Sam, Mary, Bob, Marty, Betty,

Tom, Teddy, Jerry, Ben, Sara, Donald, Damon, Darlene,

Dustin);

package EmpNameIO is new Ada.Text\_IO.Enumeration\_IO(EmpName); use EmpNameIO;

type LegalResponce is (yup, affirmative, Yes, nope, negative, No);

subtype PositiveResponce is LegalResponce range yup..Yes;

package LegalIO is new Ada.Text\_IO.Enumeration\_IO(LegalResponce); use LegalIO;

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer); use IntIO;

type Emp;

type EmpPt is access Emp;

type Emp is record

Name: EmpName;

Job: JobType;

Next: EmpPt;

end record;

procedure Free is new Ada.Unchecked\_Deallocation(Emp, EmpPt);

procedure InsertQ(NameIn: EmpName; Job: JobType;

Rear: in out EmpPt; Inserted: out Boolean) is

Pt: EmpPt;

begin

Pt := new Emp'(NameIn, Job, null);

if Pt /= null then

Inserted := true;

Rear.Next := Pt;

Rear := Pt;

else

Inserted := false;

end if;

end InsertQ;

procedure DeleteQ(NameOut: out EmpName; JobOut: out JobType;

Front, Rear: in out EmpPt; Removed: out Boolean) is

Pt: EmpPt;

begin

if Front.Next = null then

Removed := false;

else

Removed := true;

Pt := Front.Next;

Front.Next := Pt.Next;

NameOut := Pt.Name; JobOut := Pt.Job;

Free(Pt); -- Prevent memory hemmoraging.

if Front.Next = null then

Rear := Front;

end if;

end if;

end DeleteQ;

Front: EmpPt := new Emp; -- Front.Next is the actual "front" of queue.

Rear: EmpPt := Front; -- Set the queue empty initially.

OperationCompleted: Boolean;

Again: LegalResponce := affirmative;

TName: EmpName;

TJob: JobType;

begin

while (Again in PositiveResponce) loop

put("Enter name: "); get(TName); --Get emp info.

put("Enter Job type: "); get(TJob);

-- Insert in appropriate list (by job).

InsertQ(TName, TJob, Rear, OperationCompleted);

put("Enter another name (yup or nope): "); get(Again);

end loop;

while Front.Next /= null loop

new\_line; put("Employee is = ");

DeleteQ(TName, TJob, Front, Rear, OperationCompleted);

if OperationCompleted then -- This test is not really necessary.

put(Tname); put(" "); put(TJob); new\_line;

end if;

end loop;

end LinkQ;

-- Sort a group of 200 employee records by our 8 job categories. You may

-- not look at any record more than once while placing them in sorted

-- order. Print the sorted list but do not destroy the sort sequence.

with Ada.Text\_IO; use Ada.Text\_IO;

procedure LinkSort2 is

type JobType is (Pgmr, Mgr, Acct, Anal, Sales, Manuf, Inven, SoftwareEnginner);

package JobTypeIO is new Ada.Text\_IO.Enumeration\_IO(JobType); use JobTypeIO;

type EmpName is (David, Kevin, Sam, Mary, Bob, Marty, Sable, Betty,

Tom, Teddy, Jerry, Ben, Sara, Donald, Damon, Darlene,

Dustin, Desire);

package EmpNameIO is new Ada.Text\_IO.Enumeration\_IO(EmpName); use EmpNameIO;

type LegalResponce is (yup, affirmative, nope, negative);

subtype PositiveResponce is LegalResponce range yup..affirmative;

package LegalIO is new Ada.Text\_IO.Enumeration\_IO(LegalResponce); use LegalIO;

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer); use IntIO;

type Emp;

type EmpPt is access Emp;

type Emp is record Name: EmpName; Job: JobType; Next: EmpPt; end record;

SortByJob: Array(JobType) of EmpPt;

TempName: EmpName;

TempJob: JobType;

Pt: EmpPt;

Again: LegalResponce := affirmative;

begin

while (Again in PositiveResponce) loop

put("Enter name (David, Kevin, Sam, Mary, or Bob): "); get(TempName);

put("Enter Job type (Pgmr, Mgr, Acct, Anal, Sales: "); get(TempJob);

-- Insert in appropriate list (by job).

SortByJob(TempJob):= new Emp'(TempName, TempJob, SortByJob(TempJob));

put("Enter another name (yup or nope): "); get(Again);

end loop;

for I in JobType loop -- Traverse.

new\_line; put("Job Type = "); put (I); new\_line;

Pt := SortByJob(I); -- Point to first node in job list.

while Pt /= null loop

put(Pt.Name); put(" "); put(Pt.Job); new\_line; Pt := Pt.Next; -- Move down list.

end loop;

end loop;

end LinkSort2;

-- Sort a group of 200 employee records by our 8 job categories. You may

-- not look at any record more than once while placing them in sorted

-- order. Print the sorted list but do not destroy the sort sequence.

with Ada.Text\_IO; use Ada.Text\_IO;

procedure LinkSort is

type JobType is (Programmer, Manager, Accountant, Analysist,

Sales, Manufacturing, Inventory, SoftwareEnginner);

package JobTypeIO is new Ada.Text\_IO.Enumeration\_IO(JobType); use JobTypeIO;

type EmpName is (David, Kevin, Sam, Mary, Bob, Marty, Sable, Betty,

Tom, Teddy, Jerry, Ben, Sara, Donald, Damon, Darlene, Dustin, Desire);

package EmpNameIO is new Ada.Text\_IO.Enumeration\_IO(EmpName); use EmpNameIO;

type LegalResponce is (yup, affirmative, nope, negative);

subtype PositiveResponce is LegalResponce range yup..affirmative;

package LegalIO is new Ada.Text\_IO.Enumeration\_IO(LegalResponce); use LegalIO;

package IntIO is new Ada.Text\_IO.Integer\_IO(Integer); use IntIO;

type Emp is record

Name: EmpName; Job: JobType; Next: integer;

end record;

SortByJob: Array(JobType) of integer := (others => 0);

SortSpace: Array(1..200) of Emp;

Avail: integer := 1; -- Dynamic storage allocator.

Pt: integer;

Again: LegalResponce := affirmative;

begin

while (Again in PositiveResponce) loop

put("Enter name: "); get(SortSpace(Avail).Name); --Get emp info.

put("Enter Job type: "); get(SortSpace(Avail).Job);

-- Insert in appropriate list (by job).

SortSpace(Avail).Next := SortByJob(SortSpace(Avail).Job);

SortByJob(SortSpace(Avail).Job) := Avail;

-- Prepare for next dynamically allocated node.

Avail := Avail + 1;

put("Enter another name (yup or nope): "); get(Again);

end loop;

for I in JobType loop -- Traverse.

new\_line; put("Job Type = "); put (I); new\_line;

Pt := SortByJob(I); -- Point to first node in job list.

while Pt /= 0 loop

put(SortSpace(Pt).Name); put(" "); put(SortSpace(Pt).Job);

put(" link = "); put(SortSpace(Pt).Next,4); new\_line;

Pt := SortSpace(Pt).Next; -- Move down list.

end loop;

end loop;

end LinkSort;

with unchecked\_conversion; with system; use system;

with Ada.Text\_IO; use Ada.Text\_IO;

procedure uncheck2Address is

type ptr is access long\_integer;

pt: ptr:= new long\_integer'(46);

b: array(1..4) of long\_integer := (1,2,3,4);

c: address;

a: long\_integer := 1;

package long\_int\_io is new Ada.Text\_IO.Integer\_IO(long\_integer);

use long\_int\_io;

function integer\_to\_address is new unchecked\_conversion(long\_integer, address);

function address\_to\_integer is new unchecked\_conversion(address, long\_integer);

function integer\_to\_ptr is new unchecked\_Conversion(long\_integer, ptr);

begin

c := b'address;

for i in 1.. 3 loop

a := address\_to\_integer(c);

put(a); put(" ");

pt := integer\_to\_ptr(a);

put(pt.all); new\_line;

a := a + 4;

c := integer\_to\_address(a);

end loop;

end uncheck2Address;

415977414 1

415977418 2

415977422 3

--This is not the only way to do XOR etc on pointers. Package system

-- comes with the data type "address" which allows you to

-- obtain address at run time.

-- This solution utilizes "modular types" and the fact that the

-- current PC architecture is 32 bit. Since XOR is commutative,

-- m1 xor m2 xor m1 = m2 or Mary in the example. The output is

-- joe

-- Mary

-- Mary

with Ada.Text\_IO; use Ada.Text\_IO;

with unchecked\_conversion;

procedure XOREx is

type Cell;

type CellPt is access Cell;

type Cell is record

Name: String(1..4);

next: CellPt;

end record;

type Modular32 is mod 2\*\*32;

function CellPtToModular is new Unchecked\_Conversion(CellPt, Modular32);

function ModularToCellPt is new Unchecked\_Conversion(Modular32, CellPt);

p1, p2, p3: CellPt;

m1, m2, m3: Modular32;

begin

p1 := new Cell'("joe ", null);

put( p1.Name ); new\_line;

p2 := new Cell'("Mary", null);

put( p2.Name ); new\_line;

m1 := CellPtToModular(p1);

m2 := CellPtToModular(p2);

m3 := m1 xor m2 xor m1;

p3 := ModularToCellPt( m3 );

put( p3.Name );

end;