אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.
שימו לב:
על תשובות ריקות יינתן 20% מהניקוד!

בהצלחה!

תאריך הבחינה: 3.2.2014
שם המורה: ד"ר אנדרי שרף
ד"ר דוד טולפין
ד"ר מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ד
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

בסצנה שירדה בעריכה של הסרט 'מזימות בינלאומיות' (היצ'קוק, 1959), לצד סרט נע במפעל מכוניות, צועדים תוך כדי שיחה קרי גרנט ומנהל המפעל, כאשר ברקע המכונית הולכת ומורכבת, חלק אחר חלק. בסוף הדיאלוג הם מביטים במכונית, אשר הורכבה עד תום, ואומרים: זה ממש נהדר. כאשר הם פותחים את דלת המכונית, גופה נופלת החוצה.
מפעל המכוניות, והסרט, הוזכרו כמדומני בחלק מההרצאות, בשאלה זו נעסוק בתרחיש פשוט יותר: סימולציה של חציית צומת על ידי מכוניות.

מכונית מאופיינת על ידי מיקום, ויכולת התקדמות:

```
interface Car {
  Position getPosition(); // returns the position of the car
  // increase the position of the car by one. if impossible, throws CarAdvancedException
  void advance() throws CarAdvancedException;
}

interface Position {
        int getX(); // gets the horizontal position
        int getY(); // gets the vertical position
        void setX(int x); // sets the horizontal position
        void setY(int y); // sets the horizontal position
}
```

המחלקה TrafficLight מסמלצת רמזור, המחליף את צבעיו על פי מרווחי זמן בקבועים מראש.

```
enum Light {RED, YELLOW, GREEN}
```

```
class TrafficLight implements Runnable {
	public static TrafficLight create(int redInterval, int yellowInterval, int greenInterval) {
		TrafficLight tl = new TrafficLight(redInterval,yellowInterval,greenInterval);
		new Thread(tl).start();
		return tl;
	}
	private TrafficLight(int redInterval, int yellowInterval, int greenInterval) {
		_light = Light.RED;
		_redInterval = redInterval; _yellowInterval = yellowInterval; _greenInterval = greenInterval;
	}

	public Light getLight() { return _light; }

	public void run() {
		while (true){
		 try {
			Thread.sleep(_redInterval);
			_light = Light.YELLOW;
			Thread.sleep(_yellowInterval);
			_light = Light.GREEN;
			Thread.sleep(_greenInterval);
			_light = Light.RED;
		 } catch (InterruptedException e) {
			return;
		 }
		}
	}
	Light _light;
	final int _redInterval, _yellowInterval, _greenInterval;
}
```

הממשק Junction מגדיר צומת. בצומת ארבעה רמזורים, אחד לכל כיוון.
ניתן להניח כי הצומת ממוקמת תמיד ב (0,0) . הצומת והמכוניות אינם תופסים מקום. לדוגמא, מכונית מכיוון LEFT עוברת צומת מ: 1,0- ל: 1,0.
מכונית המגיעה מכיוון מסוים ומעוניינת לחצות את הצומת מצטרפת לרשימת ההמתנה עבור כיוון זה ע"י הפעלת המתודה addCar.
המתודה move מעבירה, כאשר האור מתחלף לירוק, מכוניות מרשימת ההמתנה של הכיוון הנתון אל מעבר לצומת, ע"י הפעלת מתודת ה advanced שלהם. מכונית יכולה רק להתקדם בכיוון שלה (=אין בצומת פניות ימינה/שמאלה).
המתודה מסתיימת כאשר צבע הרמזור מתחלף לאדום.

```
enum Direction {LEFT, RIGHT, UP, DOWN}
interface Junction {
 void add(Car car, Direction dir) throws AddException; // add a car to the waiting list of the given direction
 void move(Direction dir) throws MoveException; // move waiting cars from the given direction, as long as possible
}
```

א. הגדירו תכונה נשמרת ותנאי התחלה וסיום לממשק Junction. במידה ונדרש, הגדירו שאילתות נוספות בממשק לשם תיאור התכונה הנשמרת והתנאים השונים. [14 נקודות]

ב. השלימו את המימוש הבא של הממשק Junction. יש לדאוג לכך שהמחלקה תשמור על הבטיחות והנכונות עבור הרצה מקבילית (לדוגמא, מערכת המורכבת ממספר ת'רדים הפועלים מעל אובייקט SimpleJunction משותף, תוך הוספה במקביל של מכוניות להמתנה וקידום המכוניות בכיוונים שונים, ע"י הפעלת המתודות של אובייקט ה SimpleJunction המשותף).
<u>יש לאפשר מעבר במקביל של מכוניות בכיוונים שונים</u> (ימין ושמאל, או למעלה ולמטה)

ניתן לעדכן ולהוסיף מתודות ומחלקות נוספות [16 נקודות]

```
class SimpleJunction implements Junction {
 public SimpleJunction(TrafficLight leftLight, TrafficLight rightLight, TrafficLight upLight, TrafficLight downLight) {
                    _cars = new HashMap<Direction,List<Car>>();
                    _cars.put(Direction.LEFT, new LinkedList<Car>());
                    _cars.put(Direction.RIGHT, new LinkedList<Car>());
                    _cars.put(Direction.UP, new LinkedList<Car>());
                    _cars.put(Direction.DOWN, new LinkedList<Car>());
                    _lights = new HashMap<Direction,TrafficLight>();
                    _lights.put(Direction.LEFT, leftLight);
                    _lights.put(Direction.RIGHT, rightLight);
                    _lights.put(Direction.UP, upLight);
                    _lights.put(Direction.DOWN, downLight);
 }

 public void add(Car car, Direction dir) { _cars.get(dir).add(car); }

 public void move(Direction dir)  throws MoveException {
          //@TODO
 }

 Map<Direction,List<Car>> _cars;
 Map<Direction,TrafficLight> _lights;
}
```

שאלה 2 (30 נקודות)

חברת המשחקים שחררוני בע"מ החליטה להעתיק את קונספט הצומת והרכבים לעולם המשחקים. במשחק החדשני unblock-me-free יש להזיז רכבים בתוך צומת, על מנת לפנות דרך לרכב האדום לצאת. מהנדסי החברה, בוגרי בן-גוריון, החלו לממש את המשחק ב C++ אך התעייפו באמצע.

```
#include <vector>

class Cell{
public:
        Cell(){;}
        virtual ~Cell() {;}
        void SetX(int x) {_x=x;}
        void SetY(int y) {_y=y;}
        int GetX() {return _x;}
        int GetY() {return _y;}
        virtual void SetCell()=0;
        virtual void UnsetCell()=0;
private:
        int _x;
        int _y;
};

class BlockCell:public Cell {
public:
        BlockCell(){;}
        ~BlockCell(){;}
        virtual void SetCell() {_blocked=1;}
        virtual void UnsetCell(){_blocked=0;}
private:
        bool _blocked;
};

class  BlockScene{
public:
        BlockScene(int nrows, int ncols);
        BlockScene() {;}
        BlockScene(const BlockScene &other) {;}
        ~BlockScene();
        BlockScene& operator=(BlockScene &other) {return *this;}
private:
        //a 2D grid of size nrows X ncols consisting of blockcells
        std::vector< std::vector <Cell*> > _rowscols;
}
```

א. עזרו למפתחים להשלים עבור המחלקה BlockScene וממשו 1. בנאי (constructor), 2. בנאי מעתיק (copy-constructor), 3. אופרטור= 4. הורס (destructor). [12 נקודות]
הנחיות:
- יש לאתחל את כל התאים ל(0,0,false).
- בהעתקה יש לשכפל את התאים.

ב. המפתחים כתבו את קטע הקוד הבא. ציירו את תמונת הזיכרון המתקבלת עבור ריצה עד ההערה (stack, heap). [13 נקודות]

```
#include "unblock.h"
BlockScene test_dummy(BlockScene scn){
        BlockScene *scn2 = new BlockScene(4,3);
        //*****here***
        *scn2 = scn;
        return (*scn2);
}
void main(){
        BlockScene scn1(4,3);
        BlockScene *scn2 = new BlockScene(2,3);
        BlockScene scn3 = test_dummy(scn1);
}
```

ג. לגבי קטע הקוד הנ"ל (סעיף א+ב) סמן תשובה נכונה. הניחו כי המימוש בא' תקין. [5 נקודות]

1. לא מתקמפל בגלל הגדרת cell כ pure virtual
2. מתקמפל, אך תהיה בעיית זיכרון ב stack בזמן ריצה
3. מתקמפל, אך תהיה בעיית זיכרון ב heap בזמן ריצה
4. לא ניתן לבצע הקצאת משתנים על הstack ב C++ ולכן לא ברור מה יקרה
5. אף תשובה לא נכונה

שאלה 3 (30 נקודות)

קוד שרת הריאקטור, כפי שנלמד בכיתה ובתרגול, מופיע בנספח המבחן.

במהלך חודש ניסיון של בהפעלת שרת הריאקטור באתר מסוים, התלוננו לקוחות רבים כי ההתחברות לשרת לוקחת זמן רב מדי.
צוות הפיתוח של השרת העלה שתי הצעות לפתרון הבעיה:

- הגדרת שני ServerSocketChannels ב Selector של השרת
- הגדרת שני Selectors (ת'רד נפרד לכל selector, עם executor אחד משותף), עם ServerSocketChannel בכל אחד מהם.

א. ממשו את שתי ההצעות. יש לכתוב אך ורק את השינויים שלכם בקוד, תוך ציון המחלקה, המתודה, ומיקום השינוי [20 נקודות]

ב. דונו, בקיצור נמרץ, בשתי ההצעות, תוך ציון יתרונות וחסרונות [10 נקודות]

שאלה 4 (10 נקודות)

עם המעבר לתשלום נסיעות בתחבורה הציבורית בעזרת כרטיס חכם, נדרש לבנות בסיס נתונים המכיל, עבור כל לקוח, את סכומי הכסף שנותרו עבור כל סוג של נסיעה.

- קיימים סוגים שונים של מסלולי נסיעה, המאופיינים ע"י קוד, תעריף, ושם חברת האוטובוסים המפעילה קו זה.
- לקוח יכול להפקיד סכום כסף לכל סוג של מסלול נסיעה
- לקוח מאופיין ע"י שם, תעודת זהות, וטלפון (למקרה שהכרטיס אבד).

א. הגדירו מודל נתונים עבור דרישות אלו, המבוסס על טבלאות, שדות ומפתחות. [5 נקודות]
ב. נסחו שאילתת SQL, המחזירה את רשימת היתרות של 'הארפו מרקס', בקווי הנסיעה השונים של חברת 'גלי המעביר'. [5 נקודות]

נספח: קוד ה Reactor, כפי שנלמד בכיתה ובתרגול

```
public class Reactor<T> implements Runnable {

  private static final Logger logger = Logger.getLogger("edu.spl.reactor");
   private final int _port;
   private final int _poolSize;
   private final ServerProtocolFactory<T> _protocolFactory;
   private final TokenizerFactory<T> _tokenizerFactory;
   private volatile boolean _shouldRun = true;
   private ReactorData<T> _data;


  public Reactor(int port, int poolSize, ServerProtocolFactory<T> protocol, TokenizerFactory<T> tokenizer) {
      _port = port;        _poolSize = poolSize;
      _protocolFactory = protocol;        _tokenizerFactory = tokenizer;
  }

  private ServerSocketChannel createServerSocket(int port)
          throws IOException {
      try {
          ServerSocketChannel ssChannel = ServerSocketChannel.open();
          ssChannel.configureBlocking(false);
          ssChannel.socket().bind(new InetSocketAddress(port));
          return ssChannel;
      } catch (IOException e) {
          logger.info("Port " + port + " is busy");
          throw e;
      }
  }

  public void run() {
      ExecutorService executor = Executors.newFixedThreadPool(_poolSize);
      Selector selector = null;
      ServerSocketChannel ssChannel = null;

      try {
          selector = Selector.open();
          ssChannel = createServerSocket(_port);
      } catch (IOException e) {
          logger.info("cannot create the selector -- server socket is busy?");
          return;
      }

      _data = new ReactorData<T>(executor, selector, _protocolFactory, _tokenizerFactory);
```

```
ConnectionAcceptor<T> connectionAcceptor = new ConnectionAcceptor<T>( ssChannel, _data);

try {
    ssChannel.register(selector, SelectionKey.OP_ACCEPT, connectionAcceptor);
} catch (ClosedChannelException e) {
    logger.info("server channel seems to be closed!");
    return;
}

while (_shouldRun && selector.isOpen()) {
    try {
        selector.select();
    } catch (IOException e) {
        logger.info("trouble with selector: " + e.getMessage());
        continue;
    }

    Iterator<SelectionKey> it = selector.selectedKeys().iterator();
    while (it.hasNext()) {
        SelectionKey selKey = (SelectionKey) it.next();
        it.remove();

        if (selKey.isValid() && selKey.isAcceptable()) {
            logger.info("Accepting a connection");
            ConnectionAcceptor<T> acceptor = (ConnectionAcceptor<T>) selKey.attachment();
            try {
                acceptor.accept();
            } catch (IOException e) {
                logger.info("problem accepting a new connection: "
                        + e.getMessage());
            }
            continue;
        }
        if (selKey.isValid() && selKey.isReadable()) {
            ConnectionHandler<T> handler = (ConnectionHandler<T>) selKey.attachment();
            logger.info("Channel is ready for reading");
            handler.read();
        }
        if (selKey.isValid() && selKey.isWritable()) {
            ConnectionHandler<T> handler = (ConnectionHandler<T>) selKey.attachment();
            logger.info("Channel is ready for writing");
            handler.write();
        }
    }
}
```

```
    stopReactor();
}

public int getPort() {  return _port;    }

public synchronized void stopReactor() {
    if (!_shouldRun)
        return;
    _shouldRun = false;
    _data.getSelector().wakeup();
    _data.getExecutor().shutdown();
    try {
        _data.getExecutor().awaitTermination(2000, TimeUnit.MILLISECONDS);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

public static void main(String args[]) {
    if (args.length != 2) {
        System.err.println("Usage: java Reactor <port> <pool_size>");
        System.exit(1);
    }

    try {
        int port = Integer.parseInt(args[0]);
        int poolSize = Integer.parseInt(args[1]);
        Reactor<StringMessage> reactor = startEchoServer(port, poolSize);
        Thread thread = new Thread(reactor);
        thread.start();
        logger.info("Reactor is ready on port " + reactor.getPort());
        thread.join();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static Reactor<StringMessage> startEchoServer(int port, int poolSize){
    ServerProtocolFactory<StringMessage> protocolMaker = new ServerProtocolFactory<StringMessage>() {
        public AsyncServerProtocol<StringMessage> create() {
            return new EchoProtocol();
        }
    };

    final Charset charset = Charset.forName("UTF-8");
```

```
        TokenizerFactory<StringMessage> tokenizerMaker = new TokenizerFactory<StringMessage>() {
            public MessageTokenizer<StringMessage> create() {
                return new FixedSeparatorMessageTokenizer("\n", charset);
            }
        };

        Reactor<StringMessage> reactor =
                new Reactor<StringMessage>(port, poolSize, protocolMaker, tokenizerMaker);
        return reactor;
    }

    public static Reactor<HttpMessage> startHttpServer(int port, int poolSize) throws Exception{
        ServerProtocolFactory<HttpMessage> protocolMaker = new ServerProtocolFactory<HttpMessage>() {
            public AsyncServerProtocol<HttpMessage> create() {
                return new HttpProtocol();
            }
        };

        TokenizerFactory<HttpMessage> tokenizerMaker = new TokenizerFactory<HttpMessage>() {
            public MessageTokenizer<HttpMessage> create() {
                return new HttpMessageTokenizer();
            }
        };

        Reactor<HttpMessage> reactor = new Reactor<HttpMessage>(port, poolSize, protocolMaker,
tokenizerMaker);
        return reactor;
    }
}
```

```
public class ConnectionAcceptor<T> {
    protected ServerSocketChannel _ssChannel;

    protected ReactorData<T> _data;

    public ConnectionAcceptor(ServerSocketChannel ssChannel, ReactorData<T> data) {
        _ssChannel = ssChannel;        _data = data;
    }

    public void accept() throws IOException {
        SocketChannel sChannel = _ssChannel.accept();

        if (sChannel != null) {
            SocketAddress address = sChannel.socket().getRemoteSocketAddress();

            System.out.println("Accepting connection from " + address);
```

```
            sChannel.configureBlocking(false);
            SelectionKey key = sChannel.register(_data.getSelector(), 0);

            ConnectionHandler<T> handler = ConnectionHandler.create(sChannel, _data, key);
            handler.switchToReadOnlyMode();
        }
    }
}
```

```
public class ConnectionHandler<T> {

    private static final int BUFFER_SIZE = 1024;
    protected final SocketChannel _sChannel;
     protected final ReactorData<T> _data;
     protected final AsyncServerProtocol<T> _protocol;
    protected final MessageTokenizer<T> _tokenizer;
     protected Vector<ByteBuffer> _outData = new Vector<ByteBuffer>();
     protected final SelectionKey _skey;
     private static final Logger logger = Logger.getLogger("edu.spl.reactor");
     private ProtocolTask<T> _task = null;

    private ConnectionHandler(SocketChannel sChannel, ReactorData<T> data, SelectionKey key) {
        _sChannel = sChannel;        _data = data;        _skey = key;
        _protocol = _data.getProtocolMaker().create();
        _tokenizer = _data.getTokenizerMaker().create();
    }

    private void initialize() {
        _skey.attach(this);
        _task = new ProtocolTask<T>(_protocol, _tokenizer, this);
    }

    public static <T> ConnectionHandler<T> create(SocketChannel sChannel, ReactorData<T> data, SelectionKey
key) {
        ConnectionHandler<T> h = new ConnectionHandler<T>(sChannel, data, key);
        h.initialize();
        return h;
    }

    public synchronized void addOutData(ByteBuffer buf) {
        _outData.add(buf);
        switchToReadWriteMode();
    }

    private void closeConnection() {
```

```
    _skey.cancel();
    try {
        _sChannel.close();
    } catch (IOException ignored) {
        ignored = null;
    }
}

public void read() {
    if (_protocol.shouldClose())
        return;

    SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
    logger.info("Reading from " + address);

    ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
    int numBytesRead = 0;
    try {
        numBytesRead = _sChannel.read(buf);
    } catch (IOException e) {
        numBytesRead = -1;
    }
   if (numBytesRead == -1) {
        logger.info("client on " + address + " has disconnected");
        closeConnection();
        _protocol.connectionTerminated();
        return;
    }
     buf.flip();
    _task.addBytes(buf);
    _data.getExecutor().execute(_task);
}

public synchronized void write() {
    if (_outData.size() == 0) {
      switchToReadOnlyMode();
        return;
    }
    ByteBuffer buf = _outData.remove(0);
    if (buf.remaining() != 0) {
        try {
            _sChannel.write(buf);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
            if (buf.remaining() != 0) {
                _outData.add(0, buf);
            }
        }
        if (_protocol.shouldClose()) {
            switchToWriteOnlyMode();
            if (buf.remaining() == 0) {
                closeConnection();
                SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
                logger.info("disconnecting client on " + address);
            }
        }
    }

    public void switchToReadWriteMode() {
        _skey.interestOps(SelectionKey.OP_READ | SelectionKey.OP_WRITE);
        _data.getSelector().wakeup();
    }

    public void switchToReadOnlyMode() {
        _skey.interestOps(SelectionKey.OP_READ);
        _data.getSelector().wakeup();
    }

    public void switchToWriteOnlyMode() {
        _skey.interestOps(SelectionKey.OP_WRITE);
        _data.getSelector().wakeup();
    }
}
```

```
public class ProtocolTask<T> implements Runnable {

    private final ServerProtocol<T> _protocol;
    private final MessageTokenizer<T> _tokenizer;
    private final ConnectionHandler<T> _handler;

    public ProtocolTask(final ServerProtocol<T> protocol, final MessageTokenizer<T> tokenizer, final
ConnectionHandler<T> h) {
        this._protocol = protocol;
        this._tokenizer = tokenizer;
        this._handler = h;
    }

    public synchronized void run() {
      while (_tokenizer.hasMessage()) {
```

```
        T msg = _tokenizer.nextMessage();
        T response = this._protocol.processMessage(msg);
        if (response != null) {
          try {
            ByteBuffer bytes = _tokenizer.getBytesForMessage(response);
            this._handler.addOutData(bytes);
          } catch (CharacterCodingException e) { e.printStackTrace(); }
        }
      }
    }

    public void addBytes(ByteBuffer b) {
        _tokenizer.addBytes(b);
    }
}
```

```
class FixedSeparatorMessageTokenizer implements MessageTokenizer<StringMessage> {

  private final String _messageSeparator;
   private final StringBuffer _stringBuf = new StringBuffer();
   private final Vector<ByteBuffer> _buffers = new Vector<ByteBuffer>();
   private final CharsetDecoder _decoder;
   private final CharsetEncoder _encoder;

  public FixedSeparatorMessageTokenizer(String separator, Charset charset) {
    this._messageSeparator = separator;
    this._decoder = charset.newDecoder();
    this._encoder = charset.newEncoder();
  }

  public synchronized void addBytes(ByteBuffer bytes) {
      _buffers.add(bytes);
  }

  public synchronized boolean hasMessage() {
      while(_buffers.size() > 0) {
          ByteBuffer bytes = _buffers.remove(0);
          CharBuffer chars = CharBuffer.allocate(bytes.remaining());
          this._decoder.decode(bytes, chars, false);
          chars.flip();
          this._stringBuf.append(chars);
      }
      return this._stringBuf.indexOf(this._messageSeparator) > -1;
  }
```

```
  public synchronized StringMessage nextMessage() {
    String message = null;
    int messageEnd = this._stringBuf.indexOf(this._messageSeparator);
    if (messageEnd > -1) {
      message = this._stringBuf.substring(0, messageEnd);
      this._stringBuf.delete(0, messageEnd+this._messageSeparator.length());
    }
    return new StringMessage(message);
  }

  public ByteBuffer getBytesForMessage(StringMessage msg)  throws CharacterCodingException {
    StringBuilder sb = new StringBuilder(msg.getMessage());
    sb.append(this._messageSeparator);
    ByteBuffer bb = this._encoder.encode(CharBuffer.wrap(sb));
    return bb;
  }
}
```

```
public class EchoProtocol implements AsyncServerProtocol<StringMessage> {

   private boolean _shouldClose = false;
   private boolean _connectionTerminated = false;

   public StringMessage processMessage(StringMessage msg) {
      if (this._connectionTerminated) {
         return null;
      }
      if (this.isEnd(msg)) {
         this._shouldClose = true;
         return new StringMessage("Ok, bye bye");
      }
      return new StringMessage("Your message \"" + msg + "\" has been received");
   }

  public boolean isEnd(StringMessage msg) {    return msg.equals("bye");    }

  public boolean shouldClose() {   return this._shouldClose;   }

   public void connectionTerminated() {
      this._connectionTerminated = true;
   }
}
```