COURSE PROJECT E X A M P L E

KHARKOV NATIONAL UNIVERSITY OF RADIOELECTRONICS

Education Center in English

Computer Engineering Faculty

Course Project

Subject:“ Programming”

Topic: “Name of your topic”

Student:

_____

Name, group

Examined by:

PhD, Ivanisenko Igor

Kharkiv 2019

TASK LIST (Page – 1)

Put your varian No and task description

COURSE PROJECT E X A M P L E

ABSTRACT

This course project consists from ___ pages, ___pictures, ___tables, ___references, ____appendixes.

In course project we developed wireless network of traiding company with Internet connection.

The general aim of this project is planning and network development with Internet connection using wireless technologies of data transfer, calculating radio-path, losses in the channel, summary strengthening of a signal in a wireless network and a power stock of a radio channel.

INTERNET-CAFÉ, WIRELESS NETWORK, ACCESS POINT, RADIOLINE, LOCAL AREA NETWORK, INTERNET, WIRELESS ROUTER, DATA LINK

Page numbering and page sequence in course project Report

Page No

1 – title list

2– task list

4 Abstract

5 Contents (if this content is placed in one page)

On these pages no need to put page No – just take into account

6 INTRODUCTION – IN THIS PAGE YOU MUST PUT NUMBER IN RIGHT UPPER CORNER

COURSE PROJECT E X A M P L E

CONTENTS

LIST OF KEYWORDS 6
INTRODUCTION 7
1 LITERATURE OVERVIEW AND PROBLEM DEFINITION 8
1.1 Systems of fixed broadband wireless access 8
1.2 Overview of Networks Types and topologies 8
1.2.1 Problem definition 8
2 ANALYSIS OF REQUIREMENTS FOR THE NETWORK 10
2.1 General requirements for the network 10
2.2 summary 11
CONCLUSIONS 12
BIBLIOGRAPHY 13
Appendix A 14
Appendix B 15
Appendix B 14
Appendix C 15

List of abbreviations

LAN – Local Area Network

TNC – Terminal Node Controller

CSMA/CA – Carrier Sense Multiple Access / Collision Avoidance

CSMA/CD – Carrier-Sense-Multiply-Access With Collision Detection

RTS – Ready To Send

CTS – Clear To Send

BSS – Basic Service Set

ESS – Extended Service Set

PDA – Personal digital assistant

WLAN – Wireless Local Area Network

INTRODUCTION

The world is going through a transition from industrial society to the information. The main objective of the State policy of Ukraine in the field of information society is information providing socio-economic development of the State, transition to a new phase of constructing democratic information society and the country into the global information community.

1 LITERATURE OVERVIEW AND PROBLEM DEFINITION

Paragraph Field 1.25 cm

Empty line

Wireless networks, also called Wi-Fi networks, or WLAN (Wireless LAN), have clear advantages compared with embracing its traditional wired networks. First of all, the wireless network does not need cables.

Left field 2.5 cm

Right field 1.5 cm

Empty line

1.1 Systems of fixed broadband wireless access

Empty line

Analysis of the results of technology development of user access over the past decade shows that for the provision of multimedia services at the present time there is a wide choice of wireless user access. Currently, radio systems are built in accordance with the following standards [1, 2, 10]:

1. HiperLAN2;
2. IEEE 802.11/b/g/n.

Empty line

1.2 Overview of Networks Types and topologies

Empty line

One way to categorize the different types of computer network designs is by their scope or scale. For historical reasons, the networking industry refers to nearly every type of design as some kind of area network. Common examples of area network types are:

- LAN - Local Area Network,
- WLAN - Wireless Local Area Network,
- WAN - Wide Area Network,

Empty line

1.2.1 Networks topologies

Empty line

A computer network topology is the physical communication scheme used by connected devices. These pages illustrate the common computer network topologies including bus, ring and star topology diagrams. More complex networks can be built as hybrids of two or more of these basic topologies.

1 is Number of current Chapter

Figure 1.1 illustrates of existing the network topologies.

1 is Number of current figure in Chapter 1

Empty line

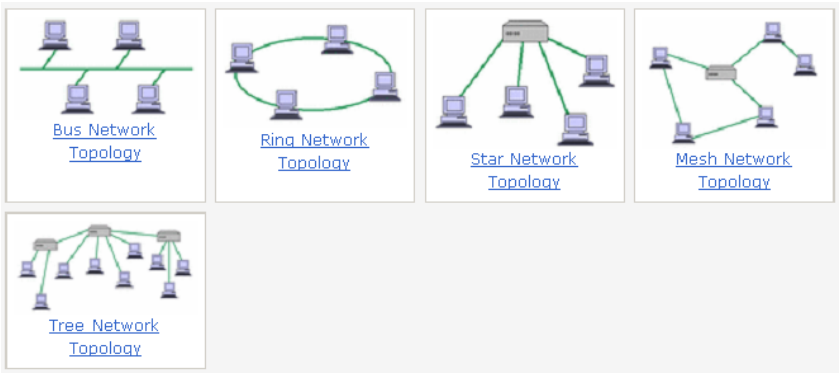

Figure 1.1 – Types of Network topologies

Center alligment

Empty line

Bus networks (not to be confused with the system bus of a computer) use a common backbone to connect all devices. A single cable, the backbone functions as a shared communication medium that devices attach or tap into with an interface connector. A device wanting to communicate with another device on the network sends a broadcast message onto the wire that all other devices see, but only the intended recipient actually accepts and processes the message.

NEXT chapter MUST be from NEW page

2 ANALYSIS OF REQUIREMENTS FOR THE NETWORK

We analyzed the requirements for the construction of wireless networks of a small business for the organization it in Internet café offices.

14pt Times New Roman Font and Justify Alligment for all text

2.1 Wireless Mesh Network

A wireless mesh network (WMN) consists of mesh nodes that form the backbone of the network. The nodes are able to configure automatically and reconfigure dynamically to maintain the mesh connectivity. This gives the mesh its "self-forming" and "self-healing" characteristics. This self-sufficient relationship between the mesh nodes removes the need for centralized management. Intelligent routing allows mesh nodes to route data packets for nodes that may not be within direct wireless range of each other. Thus information can be routed from source to destination over multiple hops. This has a potential advantage in terms of network reliability over traditional single hop networks, especially for backhaul communication.

Estimated cost of works and equipment, as well as the time required to deploy the various networks are shown in Table 2.1.

2 is Number of current Chapter

Empty line

1 is Number of current Table in Chapter 2

Table 2.1 - Parameters required for the deployment of networks

Channel type	Approximate price, $	Time for preparation and assembly
Copper cable	$300-500 on the existing sewer, otherwise 2-8 thousand dollars for 1 km	Preparation of work and strip: up to 1 month (without sewer) installation of HDSL-modem: several hours
Fiber-optic cable	$ 500-1000 to the existing sewer, or 5-10 thousand dollars per 1 km	Preparation works and laying: 2-4 months (without sewer)
Optical channel	2-4 thousand dollars per set	Pre assembly: 2-3 days; setting: 2-3 hours
Continuation of table 2.1		

Inside table you can USE 12pt or 14pt Times New Roman Font

If Table takes 2 pages you should write this

3Com Office Connect Wireless	3Com Wireless 11a/b/g PCI Adapter	Supports all three existing standards, data transmission up to 54Mbps	802.11 b/g 2,4 GHz up to 2,4835 GHz, and modulation BPSK, QPSK, 16QAM, DSSS, CKK	45

Empty line

2.2 Summary

Empty line

In end of this chapter we can make the follow conclusion:

1. It is necessary to organize the wireless access for several reasons:

- The problem is the high cost of installation.

- Problem - overcoming obstacles to the laying of the cable.

2. Broadband radio access system (especially the integrated solutions based on them) are an alternative to fiber optic; technology xDSL; radio relay lines, implementing the scheme peer-to-peer; optic communication lines in the following cases.

Conclusions

Currently the increasingly serious problem of constructing an internet-café network devided onto 3 officess – selection topology, hardware, software and data transfer environment. To solve this problem in the analysis of wireless access networks. Analysis showed a need for wireless access for several reasons:

1. The lack of telephones in remote regions.

2. The high cost of organizing wire access through natural and artificial physical barriers.

Bibliography

1. E. Tanenbaum. Computer Networks - St Petersburg, 2003 - 993 pp.
2. Semenov. Structured Cabling Systems - Moscow, 2004 - 656 pp.
3. www.Wikipedia.com

APPENDIX A

Emulation Mirrors Real-World State

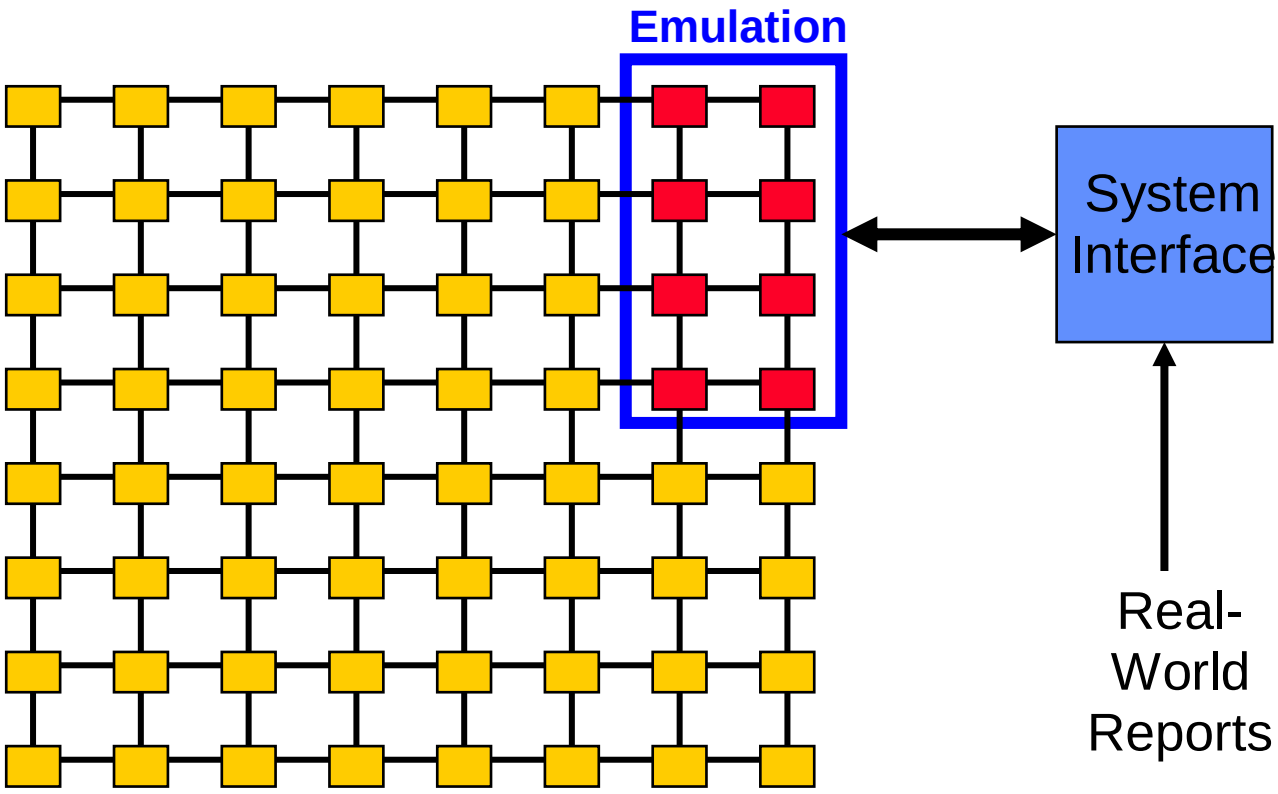

Figure A1 - Emulation Mirrors Real-World State

APPENDIX B

Program code

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, XPMan, Grids, DBGrids, DB, ADODB, StdCtrls, ExtCtrls,
  DBCtrls, Mask, Buttons;

type
  TForm1 = class(TForm)
    ADOConnection1: TADOConnection;
    ADOTable1: TADOTable;
    DataSource1: TDataSource;
    TabSheet2: TTabSheet;
    Button1: TButton;
    DBNavigator1: TDBNavigator;
    Shape1: TShape;
    Button2: TButton;
    Label1: TLabel;
    ComboBox1: TComboBox;
    procedure Edit1Change(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Edit3Change(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Button13Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses Unit2, Unit3;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
Close;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
DBNavigator1.BtnClick(nbDelete);
end;

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
ADOTable1.IndexFieldNames:=ComboBox1.Text;
DBNavigator1.BtnClick(nbRefresh);
end;

procedure TForm1.FormShow(Sender: TObject);
```

APPENDIX C

Variants for the maximum score up to 85 (from 100)

1. Write the program “calculator”. User can input two values (X and Y) and code of operation:

1 – “+” , **2** – “-“, **3** – “div”, **4** – “√ ”, **5** – $(X+Y)^2$, **6** – $(X-Y)^2$, **7** – “*”, **8** – (X+Y)/2, **9** – X+2*X + 2*Y+Y, **10** – x^y.

Computer output result and question: “Do you want to continue?”. User can answer: “Yes” or “No”.

2. Write the universal sort program. User input array that can have various size (Before user must fix size of array, then elements of array). Then computer asks: “What kind of sort you want to? Max or min”. Computer output result of sort and time that operation took.

3. Create the program for working with matrix. User input N and M (number of colons and rows) and elements of matrix. Program calculate summary of every colons and rows, then output this information on the screen of monitor (N+M values). Then program find minimum and maximum of calculated array.

4. Write program for working with numbers: User input value N ($N \in [1,50]$). If N>50, program interrupt work with exception “N is not in range”. If N is correct, computer calculated:

1. 1*2*3*4*….*N (n!)
2. 1+2+3+4+….+N
3. Fibbonachi numbers
4. (1+2)*(3+4)*(5+6)*(N+N+1)

5. Write a program - calendar. The program contains the following possibilities, which are fulfilled on demand of the user:

- Output a complete calendar for one year (2010-2030);
- Define day of week on gated in number, month, year;
- Define an amount of days in one year;
- Define an amount of days off in selected year.

6. Write the program of encryption of strings(lines). The user enters the text, then the program should encipher the text as follows: each character of string is substituted by other of English alphabet. For example, the character «a» is substituted «z». The enciphered string is output on the screen, then there is a return decryption of string. The user can change the table of the coding.

7. Write the program fulfilling the main operation on matrixes:

- Transposition of a square matrix A(n, n);
- Addition of matrixes A (n, n) and B(n, n);
- Subtraction of matrixes A (n, n) and B(n, n);
- Multiplying of matrixes A(n, n) and B(n, n);
- Finding of a maximum and minimum unit of a matrix (n, n);
- Finding of the sum of units of rows and columns;

The user enters matrixes and code of operation. The result of the operation is output on the screen of the screen monitor.

8. Write the program fulfilling the following calculations:

- $\sum_{i=1}^{N}(a_i - b_i)^K$
- $\sum_{i=1}^{N}(a_i + b_i)^K$
- $\sum_{i=1}^{N}(a_i * b_i)^K$
- $\sum_{i=1}^{N}(a_i / b_i)^K$

The user enters numbers a, b, N, K and code of operation. The result is output on the screen of the screen monitor.

9.Write the program fulfilling the following calculations

$$\sum_{i=1}^{N}(a_i - b_i)^K$$

- $$\frac{\sum_{i=1}^{N}(a_i - b_i)^K}{\sum_{i=1}^{N}(a_i + b_i)^K}$$
- $\sum_{i=1}^{N}(a_i)^K + \sum_{i=1}^{N}(b_i)^K$
- $\sum_{j}^{N}\sum_{i=1}^{N}(a_{ij} * b_i)^K$

The user enters numbers a, b, N, K and code of operation. The result is output on the screen of the screen monitor.

10. Using the std::vector class, write a program that defines an array of integers. The user should enter the

number of values in the array, and then the values themselves. After reading the values from the keyboard, print

out the partial sums. Use an *iterator* to traverse the values in the vector.

Example user input:
Enter number of elements: 4
Enter element 1: 12
Enter element 2: 3
Enter element 3: 18
Enter element 4: 7
Program output:
Partial sums: 12 15 33 40

11. Suppose we have an Employee class including a name, National Insurance number, etc. Suppose this class has two derived classes HourlyEmployee and CommissionEmployee, for people paid in different ways, and later we'll want a class for people paid both hourly and by commission. What kind of inheritance is appropriate? Sketch these classes, their fields and constructors.

12. An integer is said to be *prime* if it is divisible by only 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9

are not.

a) Write a function that determines whether a number is prime.

b) Use this function in a program that determines and prints all the prime numbers between 1 and 10,000. How many of these 10,000 numbers do you really have to test before being sure that you have found all the primes?

c) Initially, you might think that *n*/2 is the upper limit for which you must test to see whether a number is prime, but you need only go as high as the square root of *n*. Why? Rewrite the program, and run it both ways. Estimate the performance improvement.

13. An integer number is said to be a *perfect number* if the sum of its factors, including 1 (but not the number itself), is equal to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a function **perfect** that determines whether parameter **number** is a perfect number. Use this function in a program that determines and prints all the perfect numbers between 1 and 1000. Print the factors of each perfect number to confirm that the number is indeed perfect. Challenge the power of your computer by testing numbers much larger than 1000.

14. (Find the minimum value in an array) Write a recursive function recursiveMinimum that takes an integer array and the array size as arguments and returns the smallest element of the array. The function should stop processing and return when it receives an array of 1 element.

15. Write a program that prints a table of the binary, octal and hexadecimal equivalents of the decimal numbers in the range 1
through 256. If you are not familiar with these number systems

16. A mail order house sells five different products whose retail prices are product 1 — $2.98, product 2—$4.50, product 3— $9.98, product 4—$4.49 and product 5—$6.87. Write a program that reads a series of pairs of numbers as follows:
a) Product number
b) Quantity sold for one day
Your program should use a switch statement to help determine the retail price for each product. Your program should calculate and display the total retail value of all products sold last week.

17. One interesting application of computers is drawing graphs and bar charts (sometimes called "histograms"). Write a program that reads five numbers (each between 1 and 30). For each number read, your program should print a line containing that number of adjacent asterisks. For example, if your program reads the number seven, it should print *******.

APPENDIX D

Variants for the maximum score 100 (from 100)

Task 1

(*Quicksort*) As you know there are some sorting techniques of the bubble sort, bucket sort, and selection sort. We now present the recursive sorting technique called Quicksort. The basic algorithm for a single-subscripted array of values is as follows:

a) *Partitioning Step:* Take the first element of the unsorted array and determine its final location in the sorted array (i.e., all values to the left of the element in the array are less than the element, and all values to the right of the element in the array are greater than the element). We now have one element in its proper location and two unsorted subarrays.

b) *Recursive Step:* Perform step 1 on each unsorted subarray.

Each time step 1 is performed on a subarray, another element is placed in its final location of the sorted array, and two unsorted subarrays are created. When a subarray consists of one element, it must be sorted, therefore that element is in its final location.

The basic algorithm seems simple enough, but how do we determine the final position of the first element of each subarray.

As an example, consider the following set of values (the element in bold is the partitioning element—it will be placed in its final location in the sorted array):

37 2 6 4 89 8 10 12 68 45

a) Starting from the rightmost element of the array, compare each element with **37** until an element less than **37** is found.

Then swap **37** and that element. The first element less than **37** is 12, so **37** and 12 are swapped. The new array is

12 2 6 4 89 8 10 **37** 68 45

Element 12 is in italic to indicate that it was just swapped with **37**.

b) Starting from the left of the array, but beginning with the element after 12, compare each element with **37** until an element

greater than **37** is found. Then swap **37** and that element. The first element greater than **37** is 89, so **37** and 89 are

swapped. The new array is

12 2 6 4 **37** 8 10 *89* 68 45

c) Starting from the right, but beginning with the element before 89, compare each element with **37** until an element less than **37** is found. Then swap **37** and that element. The first element less than **37** is 10, so **37** and 10 are swapped. The

new array is

12 2 6 4 *10* 8 **37** 89 68 45

d) Starting from the left, but beginning with the element after 10, compare each element with **37** until an element greater than **37** is found. Then swap **37** and that element. There are no more elements greater than **37**, so when we compare **37** with itself, we know that **37** has been placed in its final location of the sorted array.

Once the partition has been applied on the array, there are two unsorted subarrays. The subarray with values less than 37 contains 12, 2, 6, 4, 10 and 8. The subarray with values greater than 37 contains 89, 68 and 45. The sort continues with both subarrays being partitioned in the same manner as the original array.

Based on the preceding discussion, write recursive function **quickSort** to sort a single-subscripted integer array. The function should receive as arguments an integer array, a starting subscript and an ending subscript. Function **partition** should be called by **quickSort** to perform the partitioning step.

Task 2.

(*Maze Traversal*) The following grid of hashes (#) and dots (**.**) is a double-subscripted array representation of a maze:

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

In the preceding double-subscripted array, the hashes (#), represent the walls of the maze and the dots represent squares in the possible paths through the maze. Moves can only be made to a location in the array that contains a dot.

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming that there is an exit). If there is not an exit, you will arrive at the starting location again. Place your right hand on the wall to your right and begin walking forward. Never remove your hand from the wall. If the maze turns to the right, you follow the wall to the right. As long as you do not remove your hand from the wall, eventually you will arrive at the exit of the maze. There may be a shorter path than the one you have taken, but you are guaranteed to get out of the maze if you follow the algorithm.

Write recursive function **mazeTraverse** to walk through the maze. The function should receive as arguments a 12-by-12 character array representing the maze and the starting location of the maze. As **mazeTraverse** attempts to locate the exit from the maze, it should place the character **X** in each square in the path. The function should display the maze after each move so the user can watch as the maze is solved.

Task 3

Write a program that uses random-number generation to create sentences. The program should use four arrays of pointers to **char** called **article**, **noun**, **verb** and **preposition**. The program should create a sentence by selecting a word at random from each array in the following order: **article**, **noun**, **verb**, **preposition**, **article** and **noun**. As each word is picked,

it should be concatenated to the previous words in an array that is large enough to hold the entire sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The program should generate 20 such sentences.

The arrays should be filled as follows: the **article** array should contain the articles **"the"**, **"a"**, **"one"**, **"some"** and **"any"**; the **noun** array should contain the nouns **"boy"**, **"girl"**, **"dog"**, **"town"** and **"car"**; the **verb** array should contain the verbs **"drove"**, **"jumped"**, **"ran"**,

"walked" and **"skipped"**; the **preposition** array should contain the prepositions **"to"**, **"from"**, **"over"**, **"under"** and **"on"**.

After the preceding program is written and working, modify the program to produce a short story consisting of several of these sentences. (How about the possibility of a random term paper writer!)

Task 4.

Write a program that encodes English language phrases into pig Latin. Pig Latin is a form of coded language often used for amusement. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following algorithm: To form a pig-Latin phrase from an English-language phrase, tokenize the phrase into words with function **strtok**. To translate each English word into a pig-Latin word, place the first letter of the English word at the end of the English word, and add the letters "**ay**." Thus the word "**jump**" becomes "**umpjay**," the word "**the**" becomes "**hetay**" and the word "**computer**" becomes "**omputercay**." Blanks between words remain as blanks. Assume that the: the English phrase consists of words separated by blanks, there are no punctuation marks and all words have two or more letters. Function **printLatinWord** should display each word. (*Hint:* Each time a token is found in a call to **strtok**, pass the token pointer to function **printLatinWord**, and print the pig Latin word.)

Task 5.

(Writing the Word Equivalent of a Check Amount) Continuing the discussion of the previous example, we reiterate the importance of designing check-writing systems to prevent alteration of check amounts. One common security method requires that the check amount be written both in numbers, and "spelled out" in words as well. Even if someone is able to alter the numerical amount of the check, it is extremely difficult to change the amount in words.

Many computerized check-writing systems do not print the amount of the check in words. Perhaps the main reason for this omission is the fact that most high-level languages used in commercial applications do not contain adequate string manipulation features. Another reason is that the logic for writing word equivalents of check amounts is somewhat involved.

Write a program that inputs a numeric check amount and writes the word equivalent of the amount. For example, the amount 112.43 should be written as

ONE HUNDRED TWELVE and 43/100

Task 6.

Create a class **Rectangle**. The class has attributes **length** and **width**, each of which defaults to 1. It has member functions that calculate the **perimeter** and the **area** of the rectangle. It has *set* and *get* functions for both **length** and **width**. The *set* functions should verify that **length** and **width** are each floating-point numbers larger than 0.0 and less than 20.0.

Task 7.

Create a class **TicTacToe** that will enable you to write a complete program to play the game of tic-tac-toe. The class contains as **private** data a 3-by-3 double array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square; place a 2 wherever the second player moves. Each move must be to an empty square. After each move, determine if the game has been won or if the game is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players automatically. Also, allow the player t o specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play threedimensional tic-tac-toe on a 4-by-4-by-4 board (Caution: This is an extremely challenging

project that could take many weeks of effort!).

Task 8.

Create a **Date** class with the following capabilities:

a) Output the date in multiple formats such as

DDD YYYY

MM/DD/YY

June 14, 2019

b) Use overloaded constructors to create **Date** objects initialized with dates of the formats in part (a).

c) Create a **Date** constructor that reads the system date using the standard library functions of the **ctime** header and sets the **Date** members.

Task 9.

Write a function template **bubbleSort** based on the sort program. Write a driver program that inputs, sorts and outputs an **int** array and a **float** array.

Task 10.

Use a single-subscripted array to solve the following problem. Read in 20 numbers, each of which is between 10 and 100, inclusive. As each number is read, print it only if it is not a duplicate of a number already read. Provide for the "worst case" in which all 20 numbers are different. Use the smallest possible array to solve this problem.

Task 11.

Write a program that simulates the rolling of two dice. The program should use **rand** to roll the first die and should use **rand** again to roll the second die. The sum of the two values should then be calculated. *Note:* Since each die can show an integer

value from 1 to 6, then the sum of the two values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums. Figure 11.1 shows the 36 possible combinations of the two dice. Your program should roll the two dice 36,000 times. Use a single-subscripted array to tally the numbers of times each possible sum appears. Print the results in a tabular format. Also, determine if the totals are reasonable (i.e., there are six ways to roll a 7, so approximately one sixth of all the rolls should be 7).

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Figure 11.1 - The 36 possible outcomes of rolling two dice

Task 12.

Write a program that runs 1000 games of craps and answers the following questions:

a) How many games are won on the 1st roll, 2nd roll, …, 20th roll, and after the 20th roll?

b) How many games are lost on the 1st roll, 2nd roll, …, 20th roll, and after the 20th roll?

c) What are the chances of winning at craps? (*Note:* You should discover that craps is one of the fairest casino games.

What do you suppose this means?)

d) What is the average length of a game of craps?

e) Do the chances of winning improve with the length of the game?

Task 13.

(*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. You have been asked to program the new system. You are to write a program to assign seats on each flight of the airline's only plane (capacity: 10 seats).

Your program should display the following menu of alternatives— **Please type 1 for "First Class"** and **Please type 2 for "Economy"**. If the person types **1**, your program should assign a seat in the first class section (seats 1-5). If the person types **2**, your program should assign a seat in the economy section (seats 6-10). Your program should print a boarding pass indicating the person's seat number and whether it is in the first class or economy section of the plane.

Use a single-subscripted array to represent the seating chart of the plane. Initialize all the elements of the array to 0 to indicate that all seats are empty. As each seat is assigned, set the corresponding elements of the array to 1 to indicate that the seat is no longer available.

Your program should, of course, never assign a seat that has already been assigned. When the first class section is full, your program should ask the person if it is acceptable to be placed in the nonsmoking section (and vice versa). If yes, then make the appropriate seat assignment. If no, then print the message **"Next flight leaves in 3 hours."**

Task 14.

Use a double-subscripted array to solve the following problem. A company has four salespeople (1 to 4) who sell five different

products (1 to 5). Once a day, each salesperson passes in a slip for each different type of product sold. Each slip contains the following:

a) The salesperson number

b) The product number

c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write a program that will read all this information for last month's sales and summarize the total sales by salesperson by product. All totals should be stored in the double-subscripted array **sales**. After processing all the information for last month, print the results in tabular format with each of the columns representing a particular salesperson and each of the rows representing a particular product. Cross total each row to get the total sales of each product for last month; cross total each column to get the total sales by salesperson for last month. Your tabular printout should include these cross totals to the right of the totaled rows and to the bottom of the totaled columns

APPENDIX E

The Requirements for Thesis Preparation

Course project has to be formatted according to the requirements in English language. It must be prepared using computer and printed. Printed thesis text, formulas, and pictures have to be completely black and clear, not blurred. The pages parameters are following:

1. Page format is A4 (210*297 mm);
2. Times New Roman font with size of 14 pt;
3. fields for each page: right offset is 1,5 cm, left – 2,5 cm, top – 2 cm, bottom – 2,5 cm;
4. paragraph indentions are 5 characters (1,25 cm);
5. line-to-line spacing – 1,5; aligment – Justify (all the thesis report)
6. formulas: general characters size – 14 pt, large index characters size – 10 pt, small index characters size – 8 pt, large characters size – 16 pt, small characters size – 12 pt;
7. intervals: distance between columns – 100%, height of upper index – 45%, depth of lower index – 25%, height of upper border – 25%.

The first page of the thesis is the title page, second page – task. Title and task pages have to be filled in according to the required form and in English and Ukrainian languages.

Reference from supervisor and the thesis review are not parts of the thesis and do not have to be included in the contents.

Each page number must be arable numeral in the right upper corner of the page. The overall pagination includes all pages, starting with the title, but the pages numbers must appear starting from the page with «List of abbreviations».

«Abstract», «Contents», «List of abbreviations», «Introduction», «Conclusions», «Bibliography», «Appendices» sections do not have to be numbered.

Each chapter of the thesis has to start from new page. Chapter content is divided into subsections and, if needed, in paragraphs and subparagraphs. All sections, chapters, subsections and other parts in the thesis must have their own title. Chapter titles must be done in capital letters and centred in the top line. The subsections, paragraphs and subparagraphs titles have to be done in lowercase letters (except the initial letter) and put in the thesis text with the paragraph indent. The words hyphenation in chapters' titles is forbidden. Points in the end of titles are not used. But if some title consists of two sentences, then they will be divided by the point.

The distance between header and plain text is two lines. The distance between two adjacent headers is 1.5 lines. Section' title cannot be put in the bottom lines of the page, if after this title only one plain text line will be arranged.

Chapters have to be numbered serially by arable numerals in the overall thesis. Point cannot be put in the end of number of a section. Subchapters have also to be numbered by arable numerals within its chapter. Each numbered unit has a

chapter number and its own number, divided by dot. For instance, subchapter 3.2 indicates the second section of the third chapter. 3.2.4 indicates fourth paragraph of subchapter 3.2. The each paragraph text begins with indent.

All illustrations included to the thesis (this can be chart, drawing or photographs) are called Figures. A figure should be placed in the thesis immediately after the first referring text. Figures must have sequential numbers as arable numerals in the range of chapter. Number of the figure includes the chapter number and personal figure number separated by dot. The figure number is put after the word «Figure» and has centred alignment under the illustration. Caption «Figure 2.3» indicates third figure of the second chapter.

Figure should have its title and, if necessary, explaining subscription text. Title of figure is put after the caption «Figure» and its number in lowercase letters (except the initial character). Subscription text is put between a figure and its title. If the figure does not fit into one page, it can be divided into two figures. The first figure caption contains figure title, subscription text is put on each page with this figure, and after figure must be located following caption: «Figure {number of figure}, page {number of figure page}». Reference to the figure includes the word «Figure» with its number.

Statistical and other data can be given in table. Tables must be numbered and have title according to the same rules as for figures numbering. The table caption is put together with the word «Table» in the left upper corner of the table. If the table has to be placed on two pages, the title line of the table has to be repeated with caption «Continuation of Table {table number}».

Formulas in the thesis must also be numbered according to the same rules as for figures and tables. All formulas must be placed in the thesis with the centre alignment. Formula’s number must be put with right border alignment on the level of formula’s lowest line. Between formulas and other text must be one empty line from each side. Explanation of the formula symbols must be situated immediately after the formula according to the order of appearing the symbols in the formula. Each new symbol explanation should be from new line. The first line of explanations begins from word «where» without a colon.

Enumerations can be situated in paragraphs and subparagraphs. It is needed to put a colon before any enumeration. A lowercase alphabet letter with parenthesis or hyphen has to be before each first level enumeration element. For other levels of enumeration one should use arable numerals with parenthesis.

Thesis text may contain small pieces of program code (printouts). Printouts must be formatted as examples, and supported with the «Example» subscription and sequential number, using the same rules as for figures.

Bibliography must be presented in the thesis according to the standard. It contains all sources which were cited by the order of appearing references in the thesis. All references must be numbered by arable numerals in parentheses.

Thesis appendices are separate sections placed after conclusions. Each appendix starts from a new page. The word «Appendix» with lowercase letters (except the initial) is put in the centre of top line. The capital alphabet letter after this word specifies each appendix, for instance, «Appendix A». Each appendix must

also have title. Appendices are included in the overall thesis page numeration. If necessary, appendix text can be divided into sections, subsections, paragraphs and subparagraphs which are numbered within each appendix. For instance, A.2 is the second section of Appendix A; C.3.1 is subsection 3.1 of Appendix C. In case of attaching figure, table or formula, it should be numbered within each appendix, for example, Figure D.5, Table E.3.

B.2 The Requirements for Thesis Graphic Documentation

Graphic documentation contains tables, graphs, charts, algorithms, time diagrams, computer images, represented according to the standard. Graphic documentation must be carried out in one of the standard graphic formats and printed on A4 sheets. Students generally use the PowerPoint software for creating this part of Bachelor degree work. Five printed copies of graphic documentation should be bounded with its title page (which must be signed by student and supervisor) and have to be given to the State examining board. The presentation size should not exceed 12 – 15 slides.