Practical Exercise 1 – Revision on Basic C++ and Algorithms

### Overall Objective

* To design algorithm and then write the code that solves a particular problem.

By the end of this practical, you should be able to:

* understand and use the Visual Studio.net integrated development environment (IDE). You can learn more about it from <http://msdn.microsoft.com/en-us/visualc/aa336395>.
* create, edit, and run your C++ programs
* know the input and output functions that you need for UECS1023.

### Background

You will need to know the topics below in order to do this practical:

|  |  |
| --- | --- |
| 1. data types, values and variables | 5. arrays |
| 2. input/output | 6. strings and strings handling library |
| 3. selection and iteration | 7. etc. |
| 4. functions |  |

### Description

**Part 1: Using Microsoft Visual Studio**

1. Download the practical material practical1.zip from WBLE.

2. Start MS Visual Studio, create an example project and code window. Copy the example program *udt.cpp* (from practical1.zip) into the code window.

In the following:

* parts (a), (b), (c) and (d) show you different ways to extract/view information from your program,
* part (e) shows you way to navigate inside your program.

3. Answer the following questions. The answer is provided for the first part of each question. If you cannot see the same output, you must have run in the wrong environment. Ask help from your practical supervisor!

1. Move cursor over the following names at column 2 and report what you can see at the tip of the cursor, i.e. fill in the blanks in the following table, column 3.

|  |  |  |
| --- | --- | --- |
|  | **Move the cursor to** | **Report what you see** |
| 1 | Person (at line 4) | struct Person |
| 2 | name (at line 5) | **char Person::name[20]** |
| 3 | money (at line 6) | **Double Person::money** |
| 4 | d1 part of the expression &d1 | **Double d1** |
| 5 | n part of the expression &n | **Person &n** |
| 6 | Judy part of Judy.money | **Person Judy** |
| 7 | money part of Judy.money | **Double Person::money** |
| 8 | ar part of ar[0].money | **Person ar[2]** |
| 9 | money part of ar[0].money | **Double Person::money** |

1. Place a break point at the left of the statement “system(“pause”);” inside the main(), i.e. click to the far left margin of the statement. Then run the program in debugging mode, click <Debug> and <Start Debugging>. When the running stops at the break point, move the cursor over the following names and report what you can see at the tip of the cursor, i.e. fill in the blanks in the following table.

|  |  |  |
| --- | --- | --- |
|  | **Move the cursor to** | **Report what you see** |
| 1 | Person (at line 4) | Person|{name=???phone=???money=???} |
| 2 | name (at line 5) | **-** |
| 3 | money (at line 6) | **-** |
| 4 | d1 part of the expression &d1 | **D1=** |
| 5 | n part of the expression &n | **n = {name=0x00d0fa78 "Juby Jones" phone=0x00d0fa8c "123456789" money=1000.8000000000000 }** |
| 6 | Judy part of Judy.money | **Judy = {name=0x00d0fa78 "Juby Jones" phone=0x00d0fa8c "123456789" money=1000.8000000000000 }** |
| 7 | money part of Judy.money | **Judy.money = 1000.8000000000000** |
| 8 | ar part of ar[0].money | **ar = 0x00d0f9cc {{name=0x00d0f9cc "ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ... phone=0x00d0f9e0 "ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ... money=...}, ...}** |
| 9 | money part of ar[0].money | **ar[0].money = 1000.0000000000000** |

1. Select the ‘Locals’ tab at the bottom left to view the locally defined variables / objects and fill in the following table.

|  |  |  |
| --- | --- | --- |
|  |  | **Report what you see** |
| 1 | How many variable names appear in this window? | 6 |
| 2 | What is the value for Judy? | +Judy {name=0x00d0fa78"Juby Jones"phone=0x00d0fa8c 123456789"money=1000.8000000000000 } Person |
| 3 | What is the value for John? | +John{name=0x00d0fa40 "ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ...phone=0x00d0fa54 "ÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌÌ... money=200.50000000000000 }Person |
| 4 | Why the values for Judy and John are different? | Because they are two different variable |
| 5 | Click on the + beside ‘n’. What you can see? | Name  phone  money |
| 6 | Click on the + beside ‘ar’. What you can see? | [0]  [1] |

(d) Select the ‘Watch1’ tab at the bottom left and fill in the following table.

|  |  |  |
| --- | --- | --- |
|  |  | **Report what you see** |
| 1 | Type d1 and then <return>at the left column of Watch1. | The value (0.5) and type (double) of d1 are displayed. |
| 2 | Type &d1 and explain what you see. | + &d1 0x00d0fac0 {0.50000000000000000} double \* |
| 3 | Type Judy | + Judy {name=0x00d0fa78 "Juby Jones" phone=0x00d0fa8c "123456789" money=1000.8000000000000 } Person |
| 4 | Type d1+d2\*7 | d1+d2\*7 4.0000000000000000 double |
| 5 | What is the difference between using <local> and <Watch1>? |  |

(e) All instructions / steps here let you navigate in your program during execution under the <Debug> mode. The needed commands can be found inside the <Debug> tab.

Put in one more break point at the line of ‘Judy.money = 100;’. Stop debugging (click <Debug>, <Stop Debugging>) and restart debugging again. Fill in the table below.

|  |  |  |
| --- | --- | --- |
|  |  | **Report what you see or guess** |
| 1 | What is the current value of Judy.money? | The uninitialized (garbage) value of  -9.255963134e+061 is displayed. |
| 2 | Execute one line by pressing <F10> once, or <Debug> + <Step over>. What is the value of Judy.money now? | money=100.00000000000000 |
| 3 | Any special observation from 1 to 2? | Money has value now |
| 4 | To continue execution until the next break point, which buttons you should press? | Shift + F11 |
| 5 | What is the functionality of ‘step into’? (Look into <Debug>) | Go into the function |
| 6 | What is the functionality of ‘step out’? | To the next breakpoint |

From now on, you can use MS Visual Studio to debug your program after your program can be compiled. The important concept is that ‘seeing is believing’. You should spend time to learn more how to use MS Visual Studio in order to be professional to debug your work.

**Part 2: Satisfying the need of input / output operations for UECS1023**

In the following, we want to cover a small but effective set of methods to satisfy our needs of input/output operations for this course. In the following, more and more requirements / options will be added on for each sub-question to enhance our capabilities to tackle I/O.

1. Create a new project ‘InputOutput’ and copy the downloaded file io.cpp onto the code window. In io.cpp, we have the following declarations.

char f[10] = {′0′, ′b′, ′2′, ′d′,′4′, ′e′, ′6′, ′f′, ′8′, ′g′};

float b[34];

double ar[2][2];

ifstream in; // Check T2 for more detail on its usage.

ofstream out; // Can also check

char s[256]; // [http://www.cplusplus.com/reference/clibrary/](http://www.cplusplus.com/reference/clibrary/cstring/strcat/)

Person p;

Copy all the input files onto the directory where you have created the io.cpp. Add statements to different sections of io.cpp to accomplish each of the following tasks:

* 1. Display the value of the seventh element of character array f on screen.

|  |
| --- |
| Print on screen |

* 1. Input a value from keyboard into element 4 of array b.

|  |
| --- |
| Read from keyboard |

* 1. Display the value of the seventh element of character array f onto a file named ‘output.txt’making use of ‘out’.

|  |
| --- |
| Print onto a file |

* 1. Display the value of the seventh element of character array f onto a file which name is determined dynamically during runtime, i.e. your program prompts user to enter the output file name from the keyboard during runtime.

|  |
| --- |
| Print onto a file |

* 1. Input unknown number of inputs from a file named ‘input.txt’ into array b making use of ‘in’. It is assumed that the file contains only numbers and the number of inputs is less than 34.

|  |
| --- |
| Read from a file |

* 1. Input unknown number of inputs from keyboard into array b. It is assumed that the number of inputs is less than 34.

|  |
| --- |
| Read from keyboard |

* 1. Open the input file ‘customers.txt’ and read the name (Phua Chu Kang) from the 1st record into ‘p’.

|  |
| --- |
| Read from a file |

**Part 3: Advanced Programming Exercise (Take Home)**

**Exercise 1**

Write a complete C++ program that prompts the user to enter a series of integers terminated by a 0. The program should then print:

* The number of integers entered (not including the 0)
* The average of the integers (with 1 decimal point of precision)
* The maximum integer
* The minimum integer

Example Output:

Enter a series of integers terminated by a 0:

85 95 75 67 0

Output:

You have entered 4 integers.

The average is 80.5

The maximum is 95

The minimum is 67

**Exercise 2**

The following program generates 50 random integers in the range 100 … 999 and stores them in an array. Next the program prints them, 10 integers to a line.

#include <iostream>

#include <cstdlib> // for random functions

#include <ctime>

#include <iomanip>

#define ARRAY\_SIZE 50 // global scope

void createData(int list[]);

void printData(int list[]);

using namespace std;

int main(void)

{

int numbers[ARRAY\_SIZE];

cout << "\n\nCreating Array Data\n";

createData(numbers);

cout << "\n\nPrinting Array Data.\n";

printData(numbers);

cout << "\n\nPrint Element in Odd Index.\n";

printOddIndex(numbers);

cout << "\n\nPrint Element with Odd Value.\n";

printOddInteger(numbers);

cout << "\n\nNumber of element with even value is: " << countEven(numbers) << endl;

system("pause");

return 0;

} /\* main \*/

void createData(int list[])

{

int i;

srand(time(NULL)); //seed random function

for (i=0; i<ARRAY\_SIZE; i++)

list[i] = rand()%900 + 100; //scaled random number

} /\* createData \*/

void printData(int list[])

{

int i;

for (i=0; i<ARRAY\_SIZE; i++)

{

if (i!=0 && i%10==0) // new line after 10 values

cout << "\n";

cout << setw(5) << list[i];

}

cout << endl;

} /\* printData \*/

Modify the program in question 1 by adding a:

(a) Function called **printOddIndex** that prints the values at the odd numbered index locations, ten to a line.

(b) Function called **printOdd** that prints the odd integers, maximum of ten to a line.

1. Function called **countEven** that returns a count of the number of even values in the array.