COMP2611 Fall 2018 Homework #2

(Due 11:55 PM, Sunday, Oct 21)

Read carefully before you start:

- This is an individual assignment; all works must be your own. You can discuss with your friends but never show your code to others.
- Start your work with given skeleton code (.asm). Add your own code under TODOs in the skeleton code. Keep other parts of the skeleton code unchanged.
- Make procedure calls with registers as specified in the skeleton, otherwise the provided procedures may not work properly.
- Zip all MIPS assembly files in a single <your_stu_id>.zip file (without the brackets).
- Submit your work via http://course.cse.ust.hk/cass/submit.html. Please make sure that your work is submitted to Comp2611 Homework2. You can read the instruction page for homework submission at http://cssystem.cse.ust.hk/UGuides/cass/student.html. Multiple submissions are allowed but only the last submission before the deadline will be graded.
- **No LATE submissions will be accepted.** Plan well and don't rush to submit in the last few minutes.
- **Your submitted program must be able to run under MARS, otherwise it will not be marked.**

Task 1: Transpose Matrix (20 marks)

Given an *m-by-n* matrix (1), its transpose matrix (2) is defined as below:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \qquad \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{pmatrix}.$$

(1) Input matrix (2) Transpose matrix

Assume the matrix in (1) is physically stored in a 1D array *originalMatrix[]* by row order,

$originMatrix[] = \{a_{11}, a_{12}, \cdots, a_{1n}, a_{21}, a_{22}, \cdots, a_{2n}, \cdots, a_{m1}, a_{m2}, \cdots, a_{mn}\}$.

The transpose matrix is constructed by copying elements from *originalMatrix[]* array, and re-arranging them into another 1D array *tranposeMatrix[]* as:

$transposeMatrix[] = \{a_{11}, a_{21}, \cdots, a_{m1}, a_{12}, a_{22}, \cdots, a_{m2}, \cdots, a_{1n}, a_{2n}, \cdots, a_{mn}\}$.

For example, given matrix $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, its *originalMatrix[]* = {1, 2, 3, 4}. The corresponding transpose matrix is $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$, and the *transposeMatrix[]* = {1, 3, 2, 4}.

Refer to the below C++ implementation, its MIPS assembly implementation is given in skeleton code *TransposeMatrix.asm*. For simplicity, the matrix to be transposed is assumed to be of size 4 x 5.

Your task is to implement *transpose()* procedure. Write your code under the TODOs.

A sample run of program in MARS:

```
The original matrix is
83 67 23 12 47
7 85 33 94 15
7 28 24 48 84
60 78 84 87 100

The transposed matrix is
83 7 7 60
67 85 28 78
23 33 24 84
12 94 48 87
47 15 84 100
```

C++ Program for your reference:

```
#include <iostream> /* cout */
#include <cstdlib> /* srand, rand */
#include <ctime> /* time */
using namespace std;

void randArray(int a[], int n) {
	srand(time(NULL)); /*random seed to init the random number generator*/
	for (int i = 0; i<n; i++)
		a[i] = rand() % 100 + 1; /*random numbers in the interval [1,100]*/
}

void transpose(int a[], int b[], int n, int col) { /*n is the size of the array a */
	int index = 0;
	for (int j = 0; j < col; j++)
		for (int i = j; i < n; i += col)
			b[index++] = a[i];
}

void print(int a[], int row, int col) {
	for (int i = 0; i < row; i++) {
		for(int j = 0; j < col; j++)
			cout << a[i*col + j] << " ";
		cout << endl;
	}
	cout << endl;
}

int main() {
	int n = 20;
	int col = 5, row = 4;// col/row in the matrix
	int originalMatrix[20];
	int transposeMatrix[20];

	randArray(originalMatrix, n);
	cout << "The original matrix is\n";
	print(originalMatrix, row, col);

	transpose(originalMatrix,transposeMatrix, n, col);
	cout << "The transposed matrix is\n";
	print(transposeMatrix, col, row);

	system("pause");
	return 0;
}
```

Task 2: Remove Duplicates from a Sorted Array (10 marks)

Given an integer array A[] with elements sorted in ascending order, the C++ function *removeDuplicates()* removes duplicated array elements and returns the counts of leftover elements.

Refer to the below C++ implementation, its MIPS assembly implementation is given in skeleton code *RemoveDuplicates.asm*. For simplicity, the matrix is assumed to be of 10 elements.

Your task is to implement *removeDuplicates()* procedure. Write your code under the TODOs.

A sample run of the program in MARS:

```
The original array is
1 8 7 4 4 5 4 4 6 8
The array after deleting duplicates is
1 4 5 6 7 8
```

C++ Program:

```
#include<iostream>
#include<cstdlib>
#include<time.h>
using namespace std;

int removeDuplicates(int a[], int n) {// A[] is sorted in ascending order already
	if (a == NULL) return 0;
	int index = 0;
	for (int i = 1; i < n; i++)
		if (a[index] != a[i])
			a[++index] = a[i];
	return index + 1;
}
void sort(int a[], int n) { /*n is the size of the array a */
	int i, j, temp;
	for (i = 0; i < n - 1; i++)
		for (j = 0; j < n - 1 - i; j++)
			if (a[j + 1] < a[j]) { /* compare the two neighbours */
				temp = a[j]; /* swap a[j] and a[j+1] */
				a[j] = a[j + 1];
				a[j + 1] = temp;
			}
}
void print(int a[], int n) {
	for (int i = 0; i < n; i++)
		cout << a[i] << " ";
	cout << endl;
}

int main(){
	int A[10];
	srand(time(NULL));
	for (int i = 0; i < 10; i++)
		A[i] = rand() % 8 + 1;

	sort(A, 10);
	cout << "The original array is" << endl;
	print(A, 10);

	int length = removeDuplicates(A, 10);
	cout << "The array after deleting duplicates is" << endl;
	print(A, length);
	return 0;
}
```

Task 3: Roman Numeral Converter (20 marks)

The numeric system represented by Roman numerals originated in ancient Rome and remained the usual way of writing numbers throughout Europe well into the Late Middle Ages.

Given a decimal integer in range [1, 3999], you can convert it to roman numerals with the knowledge of Roman radices and Roman symbol mappings.

Roman radices	**1000**	**900**	**500**	**400**	**100**	**90**	**50**	**40**	**10**	**9**	**5**	**4**	**1**
Roman symbol	M	CM	D	CD	C	XC	L	XL	X	IX	V	IV	I

For example, 2234 = 1000 + 1000 + 100 + 100 + 10 + 10 + 10 + 4,
M M C C X X X IV
therefore its corresponding Roman numeral is MMCCXXXIV.

Refer to the below C++ implementation, its MIPS assembly implementation is given in skeleton code *IntegerToRoman.asm*. For simplicity, the given integer is [1, 3999].

Your task is to implement ***intToRoman()*** **procedure. Write your code under the TODOs.**

Hint in MIPS implementation:

You may have observed the Roman symbols consist of either one or two letters. Four 1D arrays are defined in skeleton code to facilitate your coding.

```
# Roman radices
roman_radix:   .word    1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1
# Roman symbols
roman_symbol:  .asciiz  "MCMDCDCXCLXLXIXVIVI"
# Byte offset of each roman numerals in roman_symbol array
roman_start: .word 0, 1, 3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 18
# Length of each Roman symbol in roman_symbol array
roman_offset: .word 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1
```

For example, *roman_radix[5]* is 90, if you want to find its correspondingly Roman symbol, you read *roman_start[5]* (which is 7) and *roman_offset[5]* (which it 2). That guides you to read 2 consecutive bytes starting from *roman_symbol[7]*. You will get XC.

A sample run of program in MARS:

```
Please enter the Integer (range:1~3999) 2234

The Roman is MMCCXXXIV
```

C++ Program:

```
#include<iostream>
#include<string>
using namespace std;

const int roman_radix[13] = { 1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1 };
//const string roman_symbol[] = { "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X",
//"IX", "V", "IV", "I" };
const char roman_symbol[] = "MCMDCDCXCLXLXIXVIVI";
const int roman_start[13] = { 0, 1, 3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 18 };
const int roman_offset[13] = { 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1 };
char roman[40];

void intToRoman(int num) {
	int index = 0;
	for (int i = 0; num > 0; ++i) {
		int count = 0;

		//deduct from the biggest radix for any many number of times as possible
		while (num >= roman_radix[i]){
			num -= roman_radix[i];
			count++; //record the number of each roman symbol of
				 //roman_radix[i]
		}
		//write roman symbols of roman_radix[i] into roman
		for (; count > 0; --count){
			for (int j = 0; j < roman_offset[i]; j++){
				char temp = roman_symbol[roman_start[i] + j];
				roman[index++] = temp;
			}
		}
	}
}

int main(){
	int n;
	cout << "Please enter the Integer(range:1~3999)" << endl;
	cin >> n;
	intToRoman(n);
	cout << "The Roman numeral representation is" << endl;
	cout << roman << endl;
	return 0;
}
```