National University of Computer and Emerging Sciences, Lahore Campus

Course:	**Computer Programming**	**Course Code:**	**CS-103**
Program:	**BS (Computer Science)**	**Semester:**	**Spring 2018**
Duration:	**60 Minutes**	**Total Marks:**	**30**
Paper Date:	**26-Feb-18**	**Weight**	**15 %**
Section:	**All**	**Page(s):**	**5**
Exam:	**Midterm-I**	**Reg. No.**	

Instruction/Notes: You can take extra sheets for rough work but not attach with this paper.

Question 1 **(5 marks)**

Write the output of the following code segment:

```
void doSomething(int **p, int size){

    *p = new int[size];
    for (int i = 0; i < size; i++)
        (*p)[i] = i + size;

    for (int i = 0; i < size; i++)
        cout<< (*p)[i] << " ";
    cout << endl;
}
```

```
int main(){

    int rows = 3;
    int ** a = new int *[rows];

    for (int i = 0; i < rows; i++)
        doSomething(&a[i], rows+i);

    for (int i = 0; i < rows; i++)
            delete[] a[i];
    delete[] a;
    return 0;
}
```

Answer:

```
3 4 5
4 5 6 7
5 6 7 8 9
```

Question 2 (Sections A, F and G ONLY) **(10 marks)**

Given a dynamic array of pointers to dynamically allocated Student objects provide implementation for a **deallocate** function (with the prototype given below) to delete all students and the array containing the pointers. Also note that the name inside each student is also a dynamically allocated array and must be deleted.

```
struct Student {
     char * name;
     int rollNumber;
};
```

```
void deallocate(Student** stds, int size);
```

```
for(int i=0; i < size; i++) {
    delete [] stds[i]->name;
    delete stds[i];
}

delete [] stds;
```

Question 2 (Sections B, C, D and E ONLY) **(10 marks)**

`int main(){`
 	
`char ** mypets = new char*[2];`
`char * Cat = new char[30];`
`char * Dog = new char[30];`
`strcpy(Cat, "Milo a Furry Cat\n");`
`strcpy(Dog, "Courage a brave Dog \n");`
`mypets[0] = Cat;`
`mypets[1] = Dog;`
`delete[] Cat;`
`for (int i = 0; i < 2; i++)`
`  cout << mypets[i] << endl;`
`delete[] mypets;`
`mypets = nullptr;  }`
`  cout << endl;`
`  }`
`}`
`return 0;`
`}`
	Output/Error:

`void main(){`
`  int ** arr = new int*[3];`
`  int ** arr2 = new int*[3];`
`  for (int i = 0; i < 3; i++){`
`  arr[i] = nullptr;`
`  arr2[i] = nullptr;`
`  }`
`  arr[0] = new int(50); // arr[0] is pointing to an int and int is initialized to 50`
`  arr[1] = new int(60);`
`  arr2[1] = new int(40);`
`  arr2[2] = arr[1];`
`  for (int i = 0; i < 3; i++){`
`  if (arr[i] != nullptr)`
`  cout<< *arr[i] <<" ";`
`  }`
`  cout << endl;`
`  for (int i = 0; i < 3; i++){`
`  if (arr2[i] != nullptr)`
`  cout << *arr2[i] << " ";`
`  }`
`  for (int i = 0; i < 3; i++){`
`  delete arr[i];`
`  delete arr2[i];`
`  }`
`  delete[] arr;`
`  delete[] arr2;`
`  arr = nullptr;`
`  arr2 = nullptr;`
`}`	

Question 3: **(15 marks)**

Write C++ code for a function that takes a 2d-dynamic array of words as input and removes all repetitions of the same words. Make sure that there are no memory leaks in your program, and the new 2d-array should contain exactly the amount of space required to store the unique words.

Following is an example:

Input array before function call

Good
Myth
Why
Good
Psych
Myth

After function call

Good
Myth
Why
Psych

```
int strlen(char* str){
     int length = 0;
     while (str[length] != '\0') length++;
     return length;
}



bool equals(char* a,char* b){

     if(strlen(a) == strlen(b)){
          for(int i=0; i < strlen(a) ; i++){
               if(a[i] != b[i]) return false;
          }

          return true;
     }

     return false;
}


void strcpy(char* a,char*& b){

     if (b != 0) delete [] b;
     b = new char[strlen(a)+1];
     int i=0;
     for(; a[i] != '\0'; i++)
          b[i] = a[i];
     b[i] = '\0';

}
```

```
void unique(char* words[],int num,char**& final, int& finalSize ){

     char** tempwords = new char*[num];
     finalSize = 0;

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

          bool exists = false;
          for(int j=0; j < finalSize; j++){

               if( equals(words[i],tempwords[j]) ){
                    exists = true;
                    break;
               }
          }

          if (! exists){
               strcpy(words[i],tempwords[finalSize++]);
          }
     }

     final = new char*[finalSize];

     for(int i=0; i < finalSize; i++){
          strcpy(tempwords[i],final[i]);
     }

     delete [] tempwords;
}
```