Exercise 1:

// Exercise 1:

cout << "Exercise 1:\n\n";

char\* inputArray = nullptr;

inputUserSentence(inputArray); // Function for taking input sentence:

// stringLength Function.

cout << "Array Length is: " << stringLength(inputArray) << "\n\n";

int stringLength(char\* arr)

{

int length = 0;

while (\*(arr + length) != '\0') // Counts till end of array, excludes nullptr.

length++;

return length;

}

void inputUserSentence(char\*& inputArray)

{

char\* inputTemp = new char[maxSentenceSize] {'\0'}; // Intitialisation with nullptr prevents errors in single word entries.

cout << "Enter a Sentence: \n";

cin.getline(inputTemp, maxSentenceSize);

cout << endl;

int length = stringLength(inputTemp);

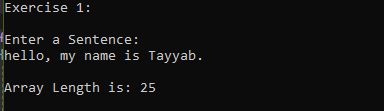
inputArray = new char[length + 1] {'\0'}; // Ensures no extra memory used for input string.

for (int i = 0; i < length; i++)

\*(inputArray + i) = \*(inputTemp + i);

delete[] inputTemp;

}



Exercise 2:

// Exercise 2:

cout << "Exercise 2:\n\n";

int size;

char\* delimiter = nullptr;

char\*\* tokens = nullptr; // Declaration of 2D Array.

inputUserDelim(delimiter); // Function for taking input of delimiter.

tokens = tokenizeString(inputArray, delimiter, size); // Tokenization function.

void inputUserDelim(char\*& delimiter)

{

char\* delimiterTemp = new char[maxDelimiterSize] {'\0'}; // Intitialisation with nullptr prevents errors in single word entries.

cout << "Enter a delimiter: \n";

if (cin.peek() == '\0') // Ensures no nullptr in buffer, and doesnt unnecessarily delete buffer.

cin.ignore();

cin.getline(delimiterTemp, maxDelimiterSize);

cout << endl;

int length = stringLength(delimiterTemp);

delimiter = new char[length + 1] {'\0'}; // Ensures no extra memory used for delimiter.

for (int i = 0; i < length; i++)

\*(delimiter + i) = \*(delimiterTemp + i);

delete[] delimiterTemp;

}

char\*\* tokenizeString(char\* inputArray, char\* delimiter, int& size)

{

int inputLength = stringLength(inputArray);

int delimiterLength = stringLength(delimiter);

int tokenSize = 0;

// Find Tokens.

for (int i = 0; i < inputLength + 1; i++) // Find number of tokens.

{

int wordLength = 0;

// Forms words from sentence, till delimiter reached.

while (inputArray[i] != '\0')

{

if (isDelimiter(inputArray, delimiter, i))

{

i += delimiterLength;

break;

}

else

wordLength++, i++; // This i++ browses the sentence.

}

// Checks if word is formed, then increments tokenSize.

if (wordLength != 0)

tokenSize++;

}

// Make 2D Array.

char\*\* tokens = new char\* [tokenSize];

tokenSize = 0; // reset so that 2D array can be traversed

for (int i = 0; i < inputLength;) // Find number of tokens.

{

char\* word = new char[maxWordSize] {'\0'}; // Initialise with null pointer, 50 is limit of characters in a word.

int wordLength = 0;

// Forms words from sentence, till delimiter reached.

while (inputArray[i] != '\0')

{

if (isDelimiter(inputArray, delimiter, i))

{

i += delimiterLength;

break;

}

else

\*(word + wordLength++) = \*(inputArray + i++); // This i++ browses the sentence.

}

// Checks if word is formed, then increments tokenSize.

if (wordLength != 0)

add2DArray(tokens, tokenSize++, word, wordLength); // increments current tokensize as well.

delete[] word;

}

// Print number of Unique Words in tokens and then the words themselves.

printTokens(tokens, tokenSize);

size = tokenSize;

return tokens;

}

bool isDelimiter(char\*& inputArray, char\*& delimiter, int index)

{

if (\*(inputArray + index) == \*(delimiter))

{

for (int j = 0; \*(delimiter + j) != '\0'; j++)

{

// If character different, delimiter not found, return false.

if (\*(inputArray + index++) != \*(delimiter + j))

return false;

}

}

else // If character different, delimiter not found, return false.

return false;

return true;

}

void add2DArray(char\*\*& tokens, const int& index, char\*& word, const int& wordLength)

{

\*(tokens + index) = new char[wordLength + 1]; // Declare new word's array.

for (int i = 0; i < wordLength + 1; i++) // Copying word into tokens.

\*(\*(tokens + index) + i) = \*(word + i);

}

void printTokens(char\*\*& tokens, const int& tokensSize)

{

cout << "There are " << tokensSize << " tokens. They are as follows:\n";

for (int i = 0; i < tokensSize; i++)

cout << i << ": " << \*(tokens + i) << endl;

cout << "\n\nReversed tokens are:\n";

for (int i = tokensSize - 1; i >= 0; i--)

cout << i << ": " << \*(tokens + i) << endl;

}

A screenshot of a computer

Description automatically generated

Exercise 3:

// Exercise 3:

cout << "\nExercise 3:\n\n";

cout << "Enter rows and columns for 2D Array: ";

int rows, cols;

cin >> rows >> cols;

int\*\* arr = create2DArray(rows, cols);

int\*\* arrShallow = shallowCopy2DArray(arr, rows, cols);

int\*\* arrDeep = deepCopy2DArray(arr, rows, cols);

// Modifying both shallow and deep copies.

cout << "\nOriginal Array is:\n";

print2DArray(arr, rows, cols);

cout << "\nModifying Shallow Copy with -1 at (0,0)\n";

\*(\*(arrShallow)) = -1;

cout << "\nOriginal:\n";

print2DArray(arr, rows, cols);

cout << "\nShallow copy:\n";

print2DArray(arrShallow, rows, cols);

cout << "\nModifying Deep Copy with 22 at (0,0)\n";

\*(\*(arrDeep)) = 22;

cout << "\nOriginal:\n";

print2DArray(arr, rows, cols);

cout << "\nDeep copy:\n";

print2DArray(arrDeep, rows, cols);

int\*\* create2DArray(int rows, int cols)

{

int\*\* arr = new int\* [rows];

for (int i = 0; i < rows; i++)

{

\*(arr + i) = new int[cols];

for (int j = 0; j < cols; j++)

{

cout << "Enter integer for (" << i << ", " << j << "): ";

cin >> \*(\*(arr + i) + j);

}

}

return arr;

}

int\*\* shallowCopy2DArray(int\*\* arr, int rows, int cols)

{

int\*\* arrShallow = new int\* [rows];

for (int i = 0; i < rows; i++)

\*(arrShallow + i) = \*(arr + i);

return arrShallow;

}

int\*\* deepCopy2DArray(int\*\* arr, int rows, int cols)

{

int\*\* arrDeep = new int\* [rows];

for (int i = 0; i < rows; i++)

{

\*(arrDeep + i) = new int[cols];

for (int j = 0; j < cols; j++)

\*(\*(arrDeep + i) + j) = \*(\*(arr + i) + j);

}

return arrDeep;

}

void print2DArray(int\*\* arr, int rows, int cols)

{

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

cout << \*(\*(arr + i) + j) << ' ';

cout << endl;

}

}

A screenshot of a computer program

Description automatically generated

Delete Functions:

// Delete Arrays:

delete[] inputArray;

inputArray = nullptr;

delete[] delimiter;

delimiter = nullptr;

delete\_2DArray(tokens, size); // Delete Function for 2D array.

delete\_2DArray(arr, rows);

delete\_2DArray(arrDeep, rows);

void delete\_2DArray(char\*\*& arr, int rows)

{

if (arr)

{

for (int i = 0; i < rows; i++)

delete[] \* (arr + i);

delete[] arr;

arr = nullptr;

}

}

void delete\_2DArray(int\*\*& arr1, int rows)

{

if (arr1)

{

for (int i = 0; i < rows; i++)

delete[] \* (arr1 + i);

delete[] arr1;

arr1 = nullptr;

}

}