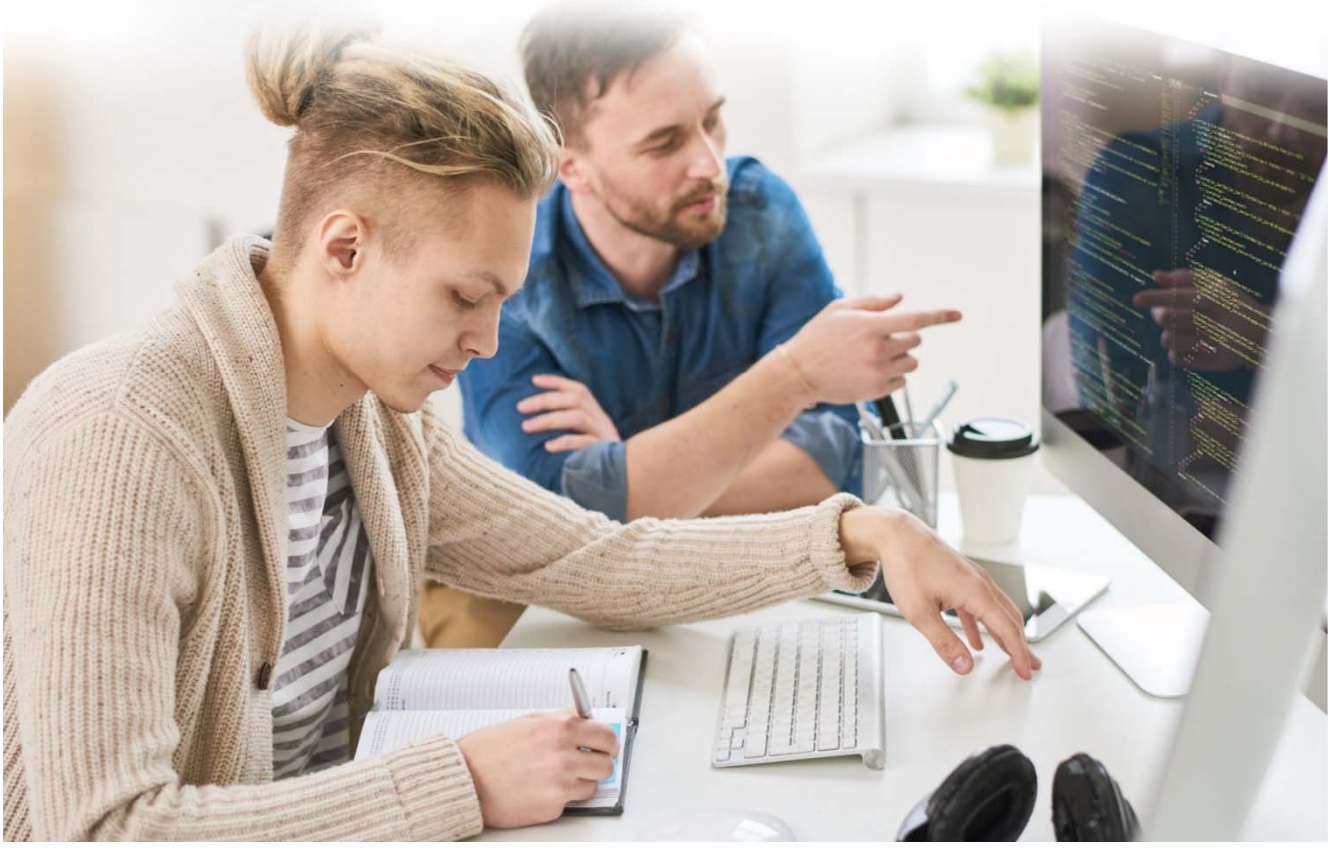
CHAPTER 8

CHARACTER AND TEXT ALGORITHM



8 Character and Text Algorithm

What is a character and text algorithm?

A character and text algorithm is an algorithm that operates on texts and characters in text.

They mainly deal with the processing of characters and strings.

Add characters to the end of text

Algorithm specification:

Purpose of the Algorithm	Add characters to the end of text
Input	characters – an array containing characters
Output	text – output text
Assumptions and remarks	N – number of characters to be added
Source directory name	ADDING-CHARACTERS-AT-END-OF-TEXT

Table 8.1. Algorithm specification

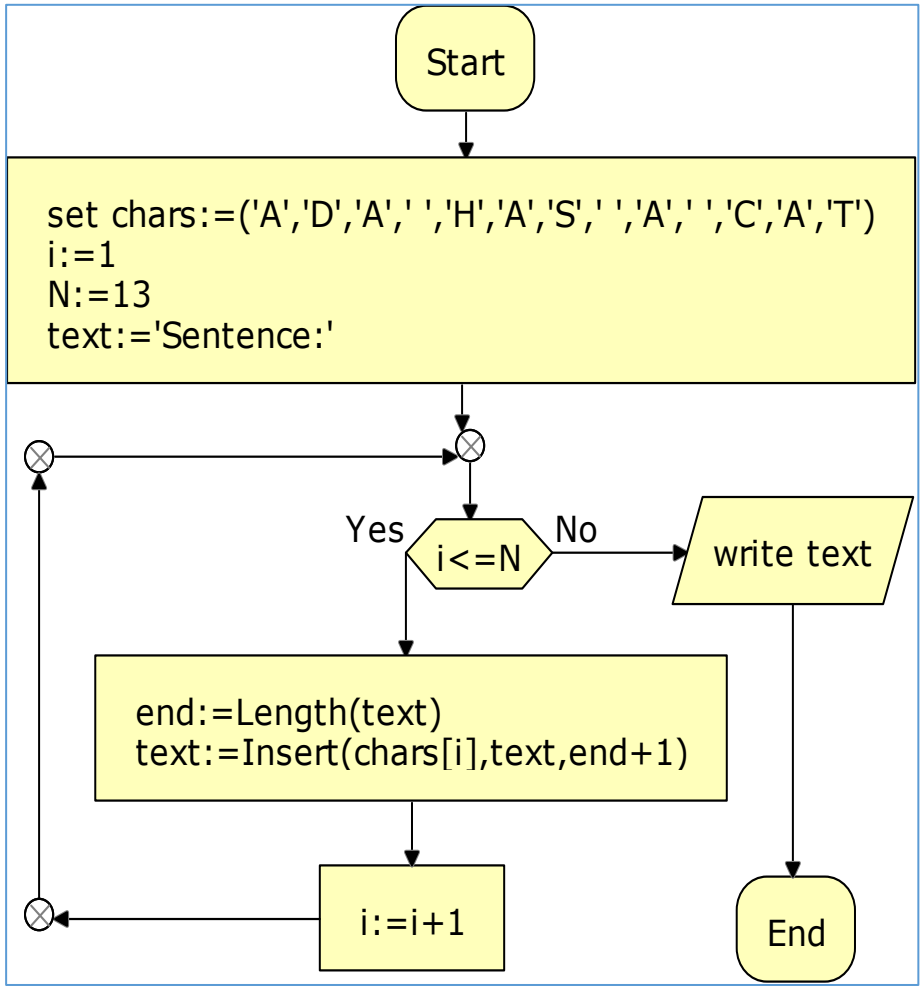


Figure 8.2. Example flowchart

```
class APPENDING_CHARACTERS_AT_THE_END_OF_TEXT
{
  private static char[] characters = {'A','D','A', ' ',
    'H','A','S',' ', 'A', ' ', 'C','A','T'};
  private static int N = characters.Length;

  static void Main(string[] args)
  {
    String text = "Sentence:";
    int i = 0;
    while (i < N)
    {
      text += characters[i];
      i++;
    }
    Console.WriteLine(text);
    Console.ReadKey();
  }
}
```

Figure 8.3. Sample code in C#

Adding characters to the beginning of text

Algorithm specification:

Purpose of the Algorithm	Adding characters to the beginning of text
Input	characters – an array containing characters
Output	text – output text
Assumptions and remarks	N – number of characters to be added
Source directory name	ADDING-CHARACTERS-AT-BEGINNING-OF-TEXT

Table 8.4. Algorithm specification

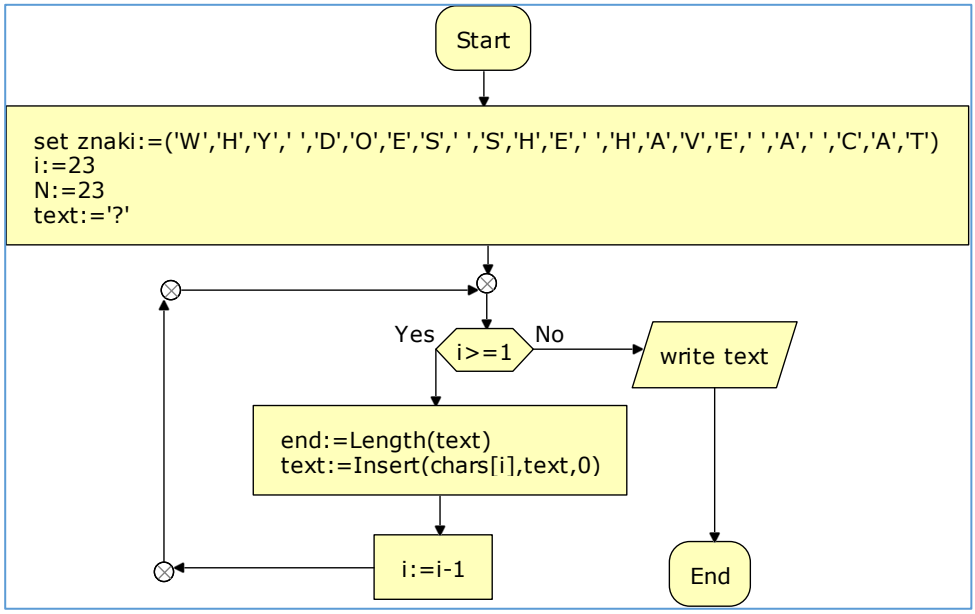


Figure 8.5. Example flowchart

```
class ADDING_CHARACTERS_AT_BEGINNING_OF_TEXT
{
  private static char[] characters = { 'W', 'H', 'Y',
    ' ', 'D', 'O', 'E', 'S',
    ' ', 'S', 'H', 'E', ' ', 'A', ' ', 'C','A','T' };
  private static int N = characters.Length;

  static void Main(string[] args)
  {
    String text = "?";
    int i = N - 1;
    while (i >= 0)
    {
      text = characters[i] + text;
      i--;
    }
    Console.WriteLine(text);
    Console.ReadKey();
  }
}
```

Figure 8.6. Sample code in C#

Alternation of uppercase and lowercase letters

The alternation of uppercase and lowercase letters consists in changing the lowercase letter to uppercase and vice versa.

Algorithm specification:

Purpose of the Algorithm	Generate text that alternates between uppercase and lowercase letters.
Input	text – text
Output	Result – text after change
Assumptions and remarks	Disadvantage: in the case of MB, the limitation is to the letters from A to Z, without Polish diacritics (23 letters and a space)
Source directory name	ALTERNATING-UPPERCASE-AND-LOWERCASE

Tabela 8.7. Specyfikacja algorytmu

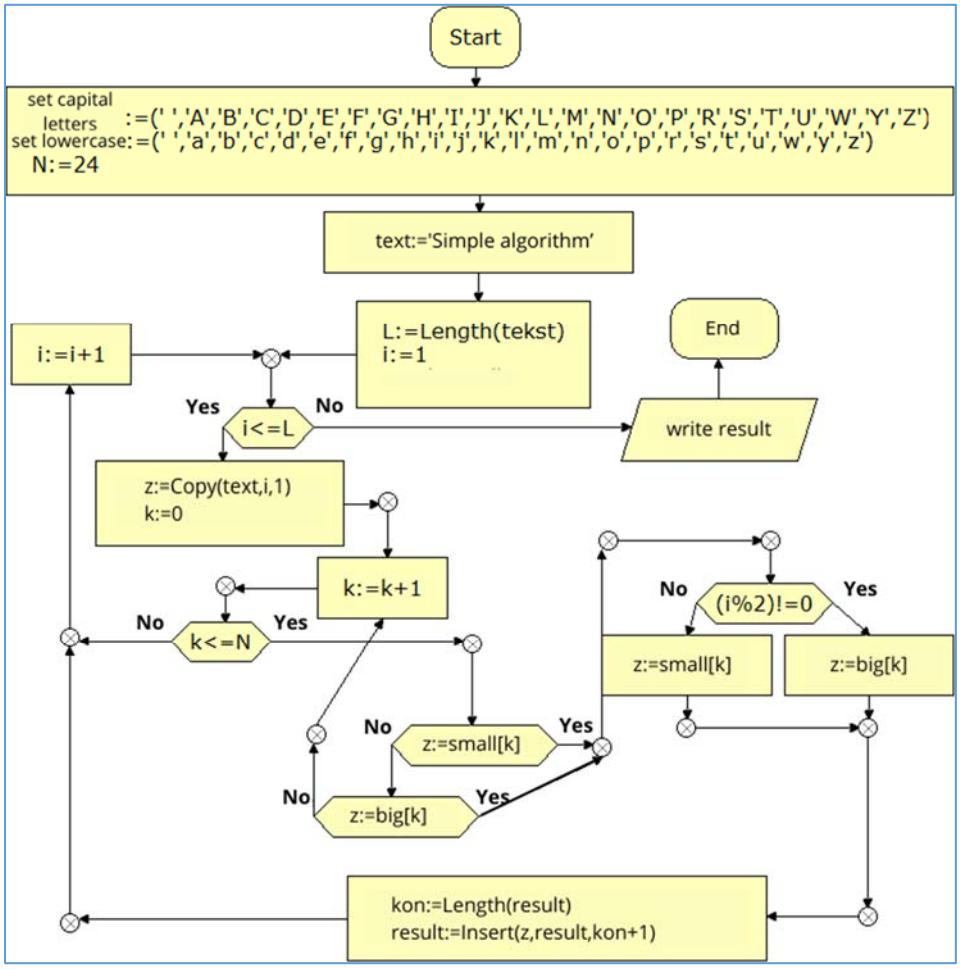


Figure 8.8. Example flowchart

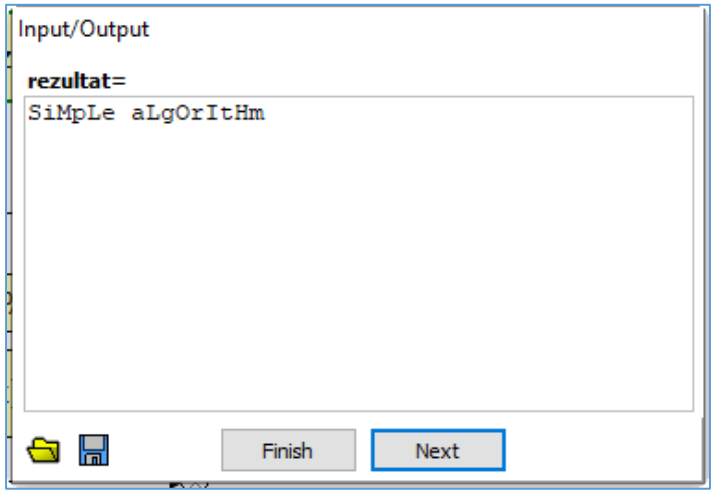


Figure 8.9. Example of the result of the algorithm

```
class ALTERNATION_UPPER_LOW_CASE
{
  static void Main(string[] args)
  {
    string text = "simple algorithm";

    string result = "";

    int count = 0;
    foreach (char c in text)
    {
      if (count % 2 == 0)
      {
        result += c.ToString().ToUpper();
      }
      else
      {
        result += c.ToString().ToLower();
      }
      count++;
    }
    Console.WriteLine(result);
    Console.ReadKey();
  }
}
```

Figure 8.10. Sample code in C#

Figure 8.11. Example of the programme's output

Naive pattern search in text

Algorithm specification:

Purpose of the Algorithm	Searching for a character pattern in text.
Input	Text – String pattern – the search string in the text
Output	pos1 – starting position pos2 – end position
Assumptions and remarks	No information if there is no pattern in the text. a, b – auxiliary variables
Source directory name	PATTERN-MATCHING

Table 8.12. Algorithm specification

Algorithm implementation:

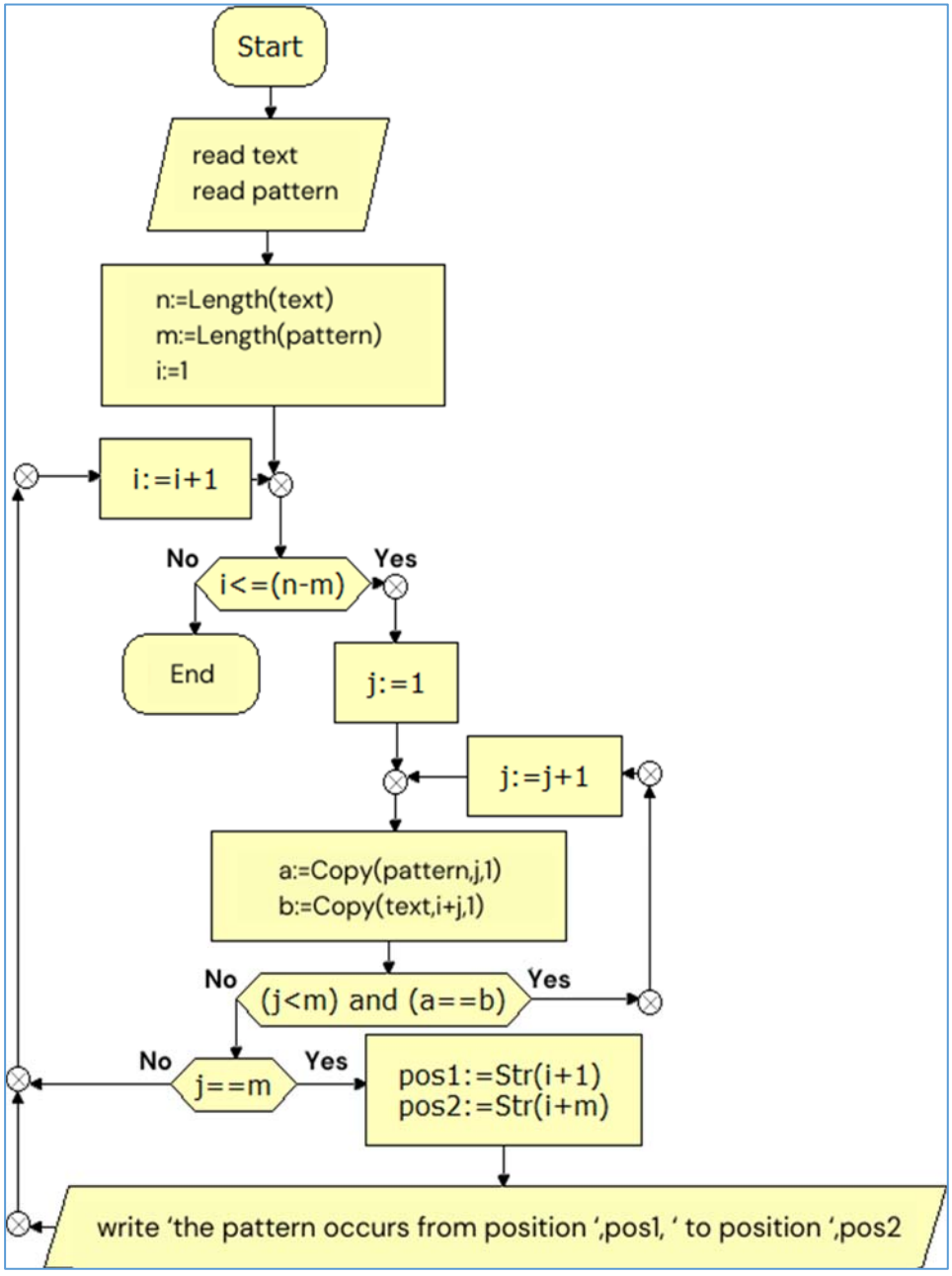


Figure 8.13. Example flowchart

```
class PATTERN_MATCHING
{
  static void Main(string[] args)
  {
    string pattern, text;
    int n, m, i = 0, j;

    Console.Write("Enter text: ");
    text = Console.ReadLine();
    Console.Write("Enter pattern: ");
    pattern = Console.ReadLine();

    n = text.Length;
    m = pattern.Length;
    while (i <= n - m)
    {
      j = 0;
      while ((j < m) && (pattern[j] == text[i + j])) j++;
      if (j == m)
        Console.WriteLine("Pattern occurs from " +
          (i + 1).ToString() + " to " +
          (i + m).ToString() + " character.");
      i++;
    }
  }
}
```

Figure 8.14. Sample code in C#

Checking if a text is a palindrome

Algorithm specification:

Purpose of the Algorithm	Check if text is a palindrome
Input	word – a string of characters
Output	Message "The specified word is a palindrome" or "The specified word is not a palindrome"
Assumptions and remarks	a, b – auxiliary variables
Source directory name	PALINDROME

Table 8.15. Algorithm specification

Algorithm implementation:

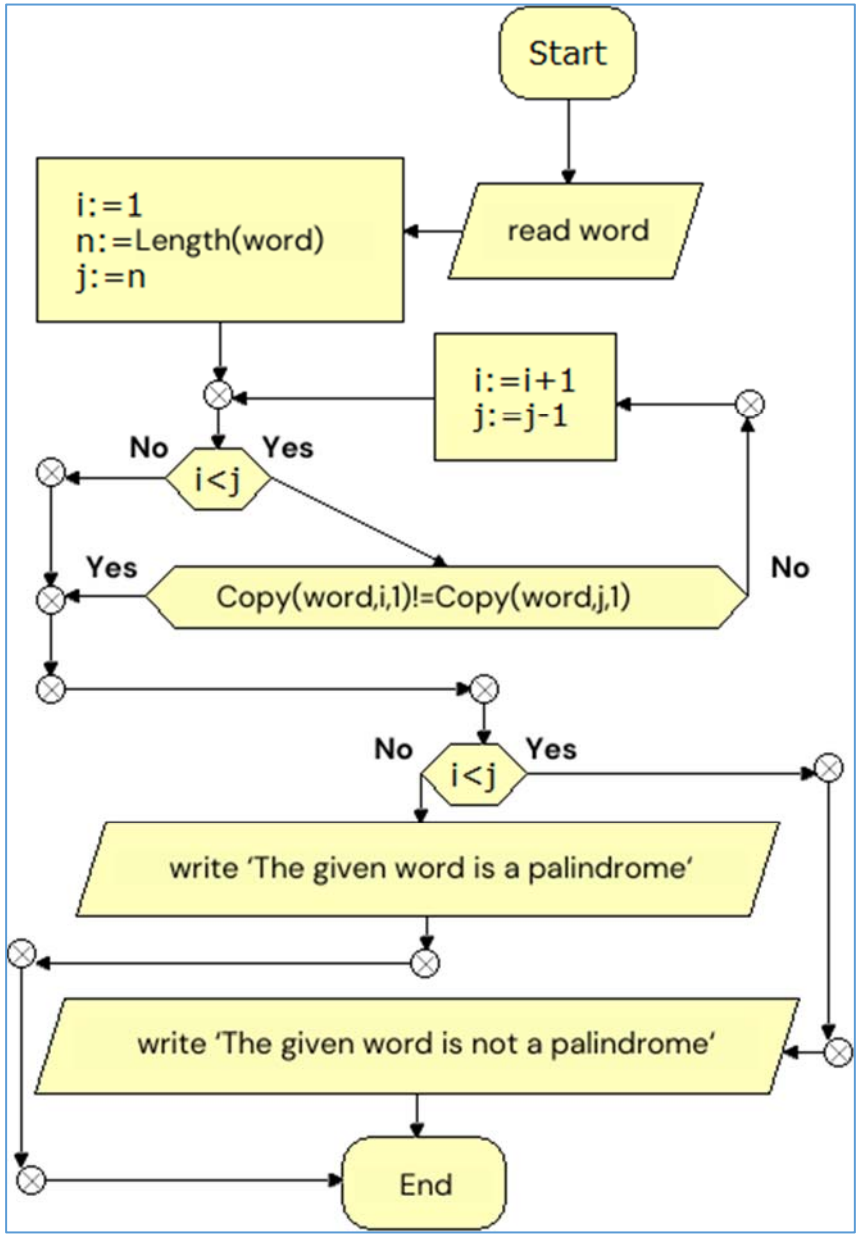


Figure 8.16. Example flowchart

```
class PALINDROME
{
  static void Main(string[] args)
  {
    int i, j;
    string expression;
    Console.WriteLine("Enter the expression you want to check: ");
    expression = Console.ReadLine();
    for (i = 0, j = expression.Length - 1; i < j; i++, j--)
    {
      if (expression[i] != expression[j]) break;
    }
    if (i < j)
      Console.WriteLine("The expression you entered is not a palindrome");
    else
      Console.WriteLine("The expression you entered is a palindrome");
  }
}
```

Figure 8.17. Sample code in C#

Checking if textes are anagrams

Algorithm specification:

Purpose of the Algorithm	Check if textes are anagrams
Input	the first – a string of characters the second – a string of characters
Output	Message "The specified textes are anagrams" or "The specified textes are not anagrams"
Assumptions and remarks	It treats both texts as arrays of signs. Uses array methods: ToLower(), ToCharArray(), Sort(), ToString() Both texts should be sorted
Source directory name	ANAGRAM

Table 8.18. Algorithm specification

Algorithm implementation:

```
function Check(first, second)
if Length(first) != Length(second)
  return False
firsttab <-- ToCharArray(ToLower(first))
secondtab <-- ToCharArray(ToLower(second))
Sort(firsttab)
Sort(secondtab)
for (index=1,...,Length(firsttab))
 {
 if firsttab[index] != secondtab[index]
   return False
 }
return True

start
read(first, second)
if check(first, second)
  write("The specified texts are anagrams")
else
  write("The specified texts are not anagrams")
stop
```

Figure 8.19. Sample pseudocode

```
class ANAGRAM
{
  public bool Check(string first, string second)
  {
    if (first.Length != second.Length)
    {
      return false;
    }
    char[] firsttab = first.ToLower().ToCharArray();
    char[] secondtab = second.ToLower().ToCharArray();
    Array.Sort(firsttab);
    Array.Sort(secondtab);
    for (int i = 0; i < firsttab.Length; i++)
    {
      if (firsttab[i].ToString() != secondtab[i].ToString())
        return false;
    }
    return true;
  }

static void Main(string[] args)
  {
  string first, second;
  Console.WriteLine("Enter the first text");
  first = Console.ReadLine();
  Console.WriteLine("Enter the second text");
  second = Console.ReadLine();
  ANAGRAM anagram = new ANAGRAM();
  if (anagram.Check(first, second) == true)
    Console.WriteLine("The given textes are anagrams");
  else
    Console.WriteLine("The given textes are not anagrams");
  Console.ReadKey();
  }
}
```

Figure 8.20. Sample code in C#