Instrukcja nr **7**	Temat: **Algorytmy kryptograficzne.**	
Przedmiot: **Języki programowania i Algorytmy**		Wymiar: LK 2x45 min.
Opracował: dr inż. Viktor Dashkiiev	Katedra Informatyki Stosowanej Wydział Mechaniczny Politechnika Krakowska	

➢ Cel zajęć

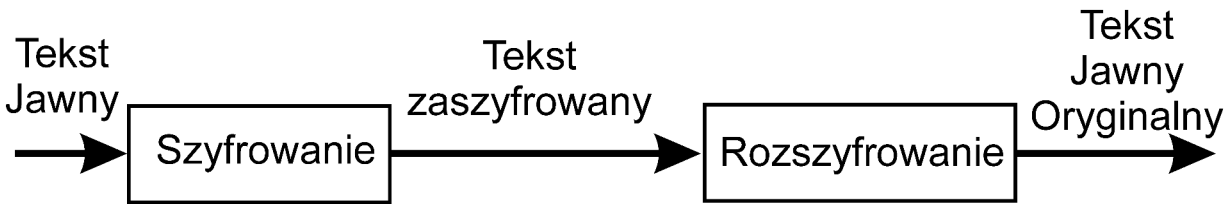
Zapoznanie się z algorytmami szyfrowania i deszyfrowania

➢ Wymagane oprogramowanie

Microsoft Visual Studio, alternatywnie - każde środowisko programistyczne umożliwiające napisanie oraz skompilowanie kodu w języku C#.

I. CZĘŚĆ TEORETYCZNA

1.1. Znaczenie i cel szyfrowania: nadawca chce wysłać wiadomość do adresata i zrobić to w bezpieczny sposób. Oznacza to, że jeśli wiadomość zostanie przechwycona, jej znaczenie nie powinno stać się znane osobom, dla których nie była przeznaczona. Osiąga się to na różne sposoby, takie jak: ukrywanie samego faktu transmisji wiadomości; transmisja wiadomości drogą radiową, we fragmentach, jednocześnie na kilku częstotliwościach, i tak dalej. Interesuje nas metoda **szyfrowania**. Oznacza to modyfikację oryginalnej wiadomości w sposób nieznany osobom postronnym. Dzięki temu nawet w przypadku, gdy cała wiadomość znajdzie się w dyspozycji osób nieupoważnionych, jej znaczenie pozostanie nieznane. Jednocześnie szyfrowanie musi mieć właściwość odwracalności: adresat, czyli odbiorca wiadomości, musi mieć możliwość całkowitego przywrócenia oryginalnej wiadomości po jej otrzymaniu. Ta odwrotna procedura nazywana jest **deszyfrowaniem**.



Oryginalna wiadomość nazywa się **jawnym tekstem** (clear text). Jawny tekst jest szyfrowany w celu ukrycia jego treści. Produktem szyfrowania jest **tekst zaszyfrowany**. Po dostarczeniu zaszyfrowanego tekstu do adresata i poddaniu go procedurze deszyfrowania, musi on zostać przekształcony w tekst jawny, ściśle identyczny z oryginałem.

W interpretacji matematycznej: jeśli M jest oryginalnym tekstem jawnym, E jest funkcją szyfrującą, to C jest tekstem zaszyfrowanym:

$$E(M)=C$$

Jeśli D jest funkcją deszyfrującą, to

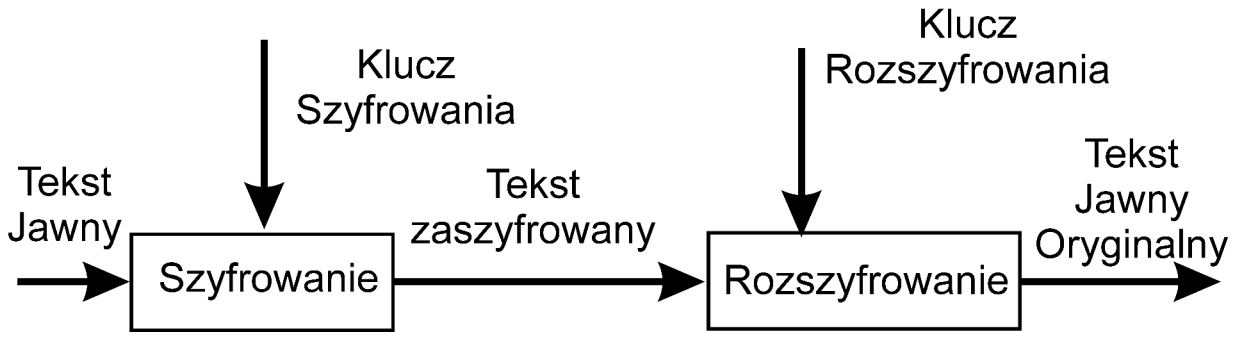
$$D(C)=M$$

Ponieważ znaczenie szyfrowania i późniejszego deszyfrowania wiadomości polega na przywróceniu oryginalnego tekstu jawnego, to w sensie matematycznym należy zapewnić następującą równość:

$$D(E(M))=M$$

Procedury szyfrowania i deszyfrowania są opisane przez tzw. **algorytm kryptograficzny**, zwany także **szyfrem**, który jest funkcją matematyczną (lub różnymi funkcjami) używaną (używanymi) do szyfrowania i deszyfrowania.
Obecnie istnieje wiele różnych algorytmów kryptograficznych. Jednak zasadniczo dzielą się one na dwie kategorie: **algorytmy bez klucza** i **algorytmy z kluczem**. Uogólniony algorytm szyfrowania bez klucza opisano powyżej. Algorytm z kluczem zakłada, że oprócz sekwencji działań służących konwersji tekstu z postaci otwartej (jawnej) do postaci zaszyfrowanej, niedostępnej dla nieuprawnionego ujawnienia jego treści, dodatkowo wykorzystywany jest określony zestaw danych, zwany **kluczem**. Szyfrowanie oparte na algorytmach jest obecnie uważane za łatwe do złamania, nie zapewnia odpowiedniej odporności szyfrowania i nie jest wykorzystywane do celów praktycznych. W algorytmach z kluczem odporność szyfrowania opiera się na założeniu, że jeśli klucz jest nieznany osobom trzecim, szyfr nie może zostać złamany.
Z kolei algorytmy szyfrowania z kluczem dzielą się na **algorytmy symetryczne** (z kluczem tajnym) i **algorytmy asymetryczne** (z kluczem jawnym).
Algorytmy symetryczne, inaczej algorytmy warunkowe (algorytm klucza tajnego, algorytm jedno-kluczowy), to algorytmy, w których klucz szyfrowania można obliczyć na podstawie klucza deszyfrującego i odwrotnie. W większości algorytmów symetrycznych klucze deszyfrowania i szyfrowania są takie same. Oni wymagają, aby przed przesłaniem bezpiecznej wiadomości nadawca i odbiorca uzgodnili klucz, który ma być użyty. Bezpieczeństwo wiadomości symetrycznej jest określane przez utrzymywanie klucza w tajemnicy. Ujawnienie klucza oznacza, że osoby trzecie mogą szyfrować i odszyfrowywać wiadomości.
Matematycznie szyfrowanie i deszyfrowanie za pomocą klucza jest opisane jako:

$$E_K(M)=C$$
$$D_K(C)=M$$



Algorytm asymetryczny lub algorytm z kluczem jawnym (publicznym) obejmuje użycie dwóch kluczy, publicznego i prywatnego (tajnego). Zazwyczaj jest to wykorzystywane w systemach dostępu współdzielonego z wieloma użytkownikami – np. w systemach rejestracji kont w sieciach społecznościowych,

bankowości elektronicznej itp. Algorytm ten wykorzystuje dwa klucze: jeden publiczny, wysyłany do wszystkich użytkowników – oraz drugi tajny, znany tylko dla konkretnego użytkownika.

1.2. Tarcza Juliusza Cezara

W tej części ćwiczenia szanownemu Państwu oferuje się programować ewentualnie najwcześniejszy algorytm szyfrowania zwany „tarczą Juliusza Cezara". Według mitu historycznego na tarczy wspomnianego bohatera starożytności zostali nałożone koncentrycznie dwa pierścienie z łacińskimi literami w otoku, z których jeden mógł się obracać względem drugiego. Obrót pierścienia o określoną liczbę pozycji umożliwiał zastąpienie liter wiadomości jawnej, wybranych na pierwszym pierścieniu, literami wiadomości zaszyfrowanej, wpisanymi na drugim pierścieniu. 2100 lat temu ten szyfr miał wystarczającą odporność. Obecnie on może mieć wartość wyłącznie edukacyjną. Jednak po opanowaniu technologii programowania, nieznacznie zmieniając kod programu, można go przekształcić w nowoczesne algorytmy, w tym te o absolutnej odporności.

Uwaga! Ze względu na pewne trudności w odtwarzaniu polskich liter „ą, ę, ż, ć, ś, ń, ł, ź" w MS Visual Studio, w tym ćwiczeniu będziemy szyfrować i deszyfrować tekst w języku angielskim. Jako tekst edukacyjny wybrano dzieło literackie Stanisława Lema „Opowieści o pilocie Pirksie" w tłumaczeniu na język angielski.

1.3. Potrójne szyfrowanie lub Algorytm z dwoma kluczami

Dowcipną odmianą algorytmu asymetrycznego jest **algorytm potrójnego szyfrowania**, czyli algorytm z dwoma kluczami prywatnymi. Nie wymaga od abonentów uzgodnienia klucza przed rozpoczęciem wymiany bezpiecznych wiadomości. Zgodnie z nim nadawca, abonent A, szyfruje wiadomość swoim własnym, tajnym kluczem K^A, znanym tylko sobie, i wysyła do adresata:

$$E_K{}^A(M)=C^A$$

Adresat, abonent B, po otrzymaniu zaszyfrowanej wiadomości C^A, szyfruje ją po raz drugi swoim własnym tajnym kluczem K^B, znanym tylko sobie. A wiadomość, już zaszyfrowana dwoma kluczami, jest odsyłana z powrotem do nadawcy:

$$E_K{}^B(C^A)=C^{A,B}$$

Nadawca odszyfrowuje otrzymaną podwójnie zaszyfrowaną wiadomość własnym kluczem:

$$D_K{}^A(C^{A,B})=C^B$$

Wiadomość, teraz zaszyfrowaną tylko kluczem odbiorcy K^B, on ponownie wysyła do odbiorcy, abonenta *B*. Odbiorca *B* odszyfrowuje odebraną wiadomość swoim własnym kluczem K^B i otrzymuje oryginalny jawny tekst:

$$D_K{}^B(C^B)=M$$

Oczywistą wadą takiego algorytmu jest konieczność wymiany zaszyfrowanych informacji **trzy razy** zamiast jednego razu, jak w innych algorytmach. Ponadto algorytm szyfrowania i klucze muszą być odwracalne: po wszystkich operacjach szyfrowania i deszyfrowania oryginalna treść musi zostać przywrócona w niezmienionej postaci. Algorytm ten okazuje się jednak niezbędny w przypadkach, gdy uzgodnienie kluczy między abonentami z tego czy innego powodu jest niemożliwe.
Korzystanie z systemów komputerowych znacznie upraszcza proces szyfrowania i deszyfrowania, ponieważ wszystkie dane w nich zawarte są już zakodowane cyfrowo. A wszystkie algorytmy szyfrowania i deszyfrowania można wykonać w postaci prostych operacji arytmetycznych: dodawania, mnożenia, przesunięcia bitowego itp.

1.4. Szyfr Vernam'a lub szyfr-notatnik

Prostą zmianą kodu programu można zamienić najstarszy i najbardziej prymitywny szyfr w szyfr całkowicie odporny na złamanie. Mówimy o tak zwanym „szyfru Vernam'a, jednorazowej podkładce", „algorytmie z szyfrem-notatnikiem". Oznacza to, że jako klucz używany jest losowy ciąg znaków (**szyfr-notatnik**). W tym przypadku każdy znak oryginalnego (otwartego) komunikatu jest zamieniany na liczbę i operację arytmetyczną (np. dodawanie) z kodem numerycznym odpowiadającego mu znaku w kluczu. (W praktyce stosowana jest nieco inna operacja **XOR - wyłączne „OR"**). Deszyfrowanie jest operacją symetryczną. Algorytm ten jest teoretycznie uważany za całkowicie odporny na złamanie – jednak ta odporność wynika z szeregu poważnych ograniczeń. Przede wszystkim klucz (sekwencja znaków w szyfr-notatniku) musi być absolutnie losowy. Oznacza to, po pierwsze, że każda taka losowa sekwencja może być użyta **tylko raz**. Jeśli ten sam klucz zostanie ponownie użyty do zaszyfrowania nowego tekstu, pojawi się wzorzec - a jeśli zostaną przechwycone dwie wiadomości zaszyfrowane tym samym kluczem, szyfr zostanie złamany. Ponadto **długość klucza musi być ściśle równa długości tekstu zaszyfrowanego**. Jeśli użyjesz klucza krótszego niż oryginalny tekst, oznacza to, że zostanie on ponownie użyty w tej samej wiadomości - a szyfr zostanie złamany nawet w przypadku przechwycenia pojedynczej wiadomości. Ponadto, podczas nadawania/odbierania zaszyfrowanej wiadomości, nie powinien zostać pominięty ani jeden znak. Jeśli jeden lub więcej znaków zostanie pominiętych, pozostałe znaki będą nie w porządku, a odbiorca nie będzie już w stanie odszyfrować wiadomości. Odwrotnie, błąd w odczytaniu

poszczególnych znaków doprowadzi do błędu w dekodowaniu tylko odpowiadających im pozycji w wiadomości - bez ingerencji w dekodowanie reszty tekstu.
W przypadkach praktycznego zastosowania szyfr-notatników (na przykład w tajnym wywiadzie) często zamiast losowej sekwencji znaków stosuje się dzieła literackie. Ściśle matematycznie spójny tekst literacki nie może być uznany za „absolutnie przypadkowy", ponieważ w każdym znanym języku litery mogą tworzyć tylko ograniczoną liczbę sekwencji. A poza tym rozszyfrowanie przynajmniej jednej wiadomości nieuchronnie doprowadzi do ujawnienia kodu - a następnie dzieła literackiego, które służy jako źródło kodu. To pozwoli odszyfrować wszystkie inne wiadomości. Jednak wymogi konspiracji pozwalają na korzystanie tylko z przedmiotów nie budzących podejrzeń, np. samych książek. rezygnując jednocześnie z absolutnej odporności kryptograficznej szyfru. Szyfr-notatniki wykorzystujące utwory literackie były używane na przykład przez sowiecki wywiad strategiczny w Niemczech i Szwajcarii podczas II wojny światowej (grupy Czerwona Kapella (Rote Kapella) i Czerwona Trójka (Rote Drei)). Pomimo wspomnianych mankamentów tego algorytmu, do dziś nikomu nie udało się rozszyfrować wielu przechwyconych wiadomości od tych grup wywiadowczych. W sieci wywiadowczej sowieckiego wywiadu w Japonii (grupa Ramsaya, inaczej grupa Richarda Sorge'a) zamiast dzieł literackich używano podręczników statystycznych jako notatników szyfrów. W związku z tym zamiast liter do zaszyfrowania tekstu użyto liczb danych statystycznych. A z teoretycznego punktu widzenia takie klucze można już uznać za „absolutnie losowe", a szyfry za „absolutnie bezpieczne". Japońskiemu kontrwywiadowi udało się rozszyfrować wiadomości dzięki temu, że wszyscy członkowie tej grupy zostali aresztowani, a wszystkie księgi szyfrów przechwycone w stanie nienaruszonym.

2. Zadania do pracy samodzielnej

2.1. Szyfrowanie wg „Tarczy Juliusza Cezara"

2.1.1. Stwórz kod programowy, jaki:
- czyta nazwę pliku tekstowego źródłowego do zaszyfrowania, napisaną we formularze Windows Forms;
- czyta liczbę-klucz do szyfrowania
- czyta ww plik tekstowy i przekształca wiersz tekstowy (String) na tablicę znaków (Chars Array);
- przetwarza każdy znak na określoną liczbę (albo przez metodę standardową C#, albo przez własną tabelę szyfrowania).
- do każdej liczby dodaje ww. klucz do zaszyfrowania (na przykład liczbę „5" lub jakąś inną);
- przetwarza otrzymaną tablicę liczb zaszyfrowaną na tekst zaszyfrowany;

- zapisuje tekst zaszyfrowany do nowego pliku pod tytułem, na przykład, „tekst_zaszyfrowany.txt”, który użytkownik ma napisać w menu dialogowym Windows Forms;
- otwiera plik z tekstem zaszyfrowanym i przekształca wiersz tekstowy (String) na tablicę znaków (Chars Array);
- przetwarza każdy znak na określoną liczbę jak ww.;
Uwaga! Zalecane jest zamienić spację w tekście pierwotnym, zaszyfrowanym lub odszyfrowanym przez inny znak, na przykład „$”, „%”, „_” lub jakiś inny
- od każdej liczby odejmuje wartość klucza do zaszyfrowania;
- przetwarza otrzymaną tablicę liczb na tekst odszyfrowany;
- zapisuje tekst odszyfrowany do nowego pliku pod tytułem, na przykład, „tekst_odszyfrowany.txt”, który użytkownik ma napisać w menu dialogowym Windows Forms;
- otwiera pierwotny tekst do zaszyfrowania i porównuje każdy kolejny znak w tekście pierwotnym i wynikowym;
- w przypadku rozbieżności sygnalizuje o tym do użytkownika;
- wymogi do interfejsu: w trybie dialogowym w utworzonym formularze WindowsForms program ma informować użytkownika o konieczności wpisania tytułów plików pierwotnego do zaszyfrowania, z tekstem zaszyfrowanym, z tekstem odszyfrowanym; wyświetlać przebieg wykonania pracy, to znaczy wykonywany etap, numer przetwarzanej pozycji w kolejce, wartość przetwarzanego znaku lub liczby i wartość otrzymaną po przetwarzaniu. Na ostatnim etapie we wypadku rozbieżności między symbolami w tekście pierwotnym i wynikowym – sygnalizować przez migające kółko czerwone i odpowiedni napis na panelu użytkownika.
2.1.2 Przygotuj tekst źródłowy do zaszyfrowania:
2.1.2.1. W folderze z materiałami tej pracy laboratoryjnej znajdź plik „Stanislav Lem_Pilot Pirx stories.txt”, kliknij go prawym przyciskiem myszy i wybierz: „otwórzyć w LibreOffice”;
2.1.2.2. W edytorze tekstu LibreOffice zaznacz dowolny fragment tekstu, według uznania własnego, skopiuj go i wklej do nowo utworzonego pliku. Zalecane jest dla ułatwienia ćwiczenia żeby wybrany fragment zawierał około 2000...2048 znaków (**dokładnie mniej niż 2049 znaków!!!**) Liczba znaków wyświetlana jest na pasku w lewym dolnym rogu okna edytora tekstu LibreOffice.
Uwaga! To jest ważne! Tekst przygotowany do zaszyfrowania musi zawierać **JEDEN AKAPIT!!!** W przeciwnym razie może wyniknąć błąd! (*przetwarzanie akapitów w tekście wymaga operacji dodatkowych*)
2.1.2.3. Wybrany do zaszyfrowania fragment tekstu zapisz pod nazwą na przykład „**Tekst_do_Zaszyfrowania.txt**” pod adresem: „**C:\studenci\mojenazwisko**”. W miejsce „mojenazwisko” wpisz imię i nazwisko studenta wykonującego pracę. Zastąp także „mojenazwisko” nazwiskiem studenta w adresach plików w kodzie programu;

2.1.2.4. Wykonaj polecenie „Build”. Jeśli nie ma błędów kompilacji, wyślij program do wykonania. Jeśli występują błędy, znajdź je i napraw, a następnie ponownie skompiluj;
2.1.2.5. W przypadku pomyślnego zakończenia programu nowo odszyfrowany tekst źródłowy ma być zapisany do pliku „**Tekst_Po_Rozszyfrowaniu.txt**” pod adresem podanym w kodzie programu.
Uwaga! Podczas dodawania, a tym bardziej przy mnożeniu liczb należy uważać, aby nie wybrać zbyt dużych wartości współczynników. W przeciwnym razie ilość pamięci zarezerwowanej dla zmiennych może zostać przekroczona, a **program wygeneruje błąd**.

2.2. Szyfrowanie wg algorytmu z dwoma kluczami

2.2.1. Zapisz gotowy kod programowy i pliki tekstów z pkt. 2.1. żeby mógł ich podać we sprawozdaniu
2.2.2. Modyfikuj kod programowy z pkt. 2.1 (***pod nowym tytułem***) tak żeby program wykonywał:
- szyfrowanie tekstu za jednym kluczem;
- zapis pliku z tekstem po pierwszym zaszyfrowaniu;
- czytanie pliku z tekstem po pierwszym zaszyfrowaniu i szyfrowanie jego drugim kluczem;
- zapis pliku z tekstem po drugim zaszyfrowaniu;
- czytanie pliku z tekstem po drugim zaszyfrowaniu i odszyfrowanie jego pierwszym kluczem;
- zapis pliku z tekstem po pierwszym odszyfrowaniu;
- czytanie pliku z tekstem po pierwszym zaszyfrowaniu i odszyfrowanie jego drugim kluczem;
- zapis pliku z tekstem po drugim odszyfrowaniu;
- porównanie tekstu pierwotnego i odszyfrowanego;
- reszta jak w pkt. 2.1.

2.3. Szyfrowanie szyfr-notatnikiem

2.3.1. Zapisz gotowy kod programowy i pliki tekstów z pkt. 2.1. żeby mógł ich podać we sprawozdaniu
2.3.2. Zmień kod programu z pkt. 2.1. tak, aby wykonywał szyfrowanie i deszyfrowanie tekstu źródłowego za pomocą szyfr-notatnika. Jako szyfr-notatnik oferowane jest inne dzieło literackie Stanisława Lema „Cyberiada”, również przetłumaczone na język angielski i znajdujące się w pliku „**Stanisław Lem The Cyberiad fables.txt**”
2.3.2.1. Wyodrębnij fragment tekstu z pliku „**Stanislav Lem The Cyberiad fables.txt**” przeznaczony do wykorzystania jako szyfr-notatnik i przetwórz go w

edytorze tekstu tak samo jak źródłowy plik tekstowy. Zapisz pod dowolną nazwą, według uznania własnego, napisaną przez użytkownika w oknie dialogowym.
2.3.2.2. Dokonaj zmian w kodzie źródłowym programu, aby program przetworzył tekst w pliku szyfr-notatnika na tablicę liczb. I dalej w trybie szyfrowania lub odszyfrowania zamiast stałej wartości klucza wg pkt 2.1 (na przykład „5") program odpowiednio dodawał lub odejmował wartość liczbową tego znaku, numer którego w kolejce odpowiada numeru znaku w tekście do szyfrowania lub odszyfrowania
Reszta jak ww.

3. Listing: przykłady kodów programowych

Przykład metody standardowej C# do przetwarzania wiersza na tablicę liczb:

```
string nazwisko = 'Kowalski';
byte[] bytes = Encoding.ASCII.GetBytes(nazwisko);

foreach(byte b in bytes)

{

// Pański kod

}
```

Przykład tabeli do szyfrowania:

a	01
b	02
z	26
itp	

Przykład kodu do czytania z pliku tekstowego:

```
//Read a Text File
using System;
using System.IO;
namespace readwriteapp
{
   class Class1
   {
      [STAThread]
      static void Main(string[] args)
      {
         String line;
         try
         {
            //Pass the file path and file name to the StreamReader constructor
```

```
            StreamReader sr = new StreamReader("C:\\Sample.txt");
            //Read the first line of text
            line = sr.ReadLine();
            //Continue to read until you reach end of file
            while (line != null)
            {
                //write the line to console window
                Console.WriteLine(line);
                //Read the next line
                line = sr.ReadLine();
            }
            //close the file
            sr.Close();
            Console.ReadLine();
        }
        catch (Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
        }
        finally
        {
            Console.WriteLine("Executing finally block.");
        }
    }
  }
}
```

Przykład kodu do zapisu w plik tekstowy:

```
//Write a text file - Version-1
using System;
using System.IO;
namespace readwriteapp
{
   class Class1
   {
      [STAThread]
      static void Main(string[] args)
      {
         try
         {
            //Pass the filepath and filename to the StreamWriter Constructor
            StreamWriter sw = new StreamWriter("C:\\Test.txt");
            //Write a line of text
            sw.WriteLine("Hello World!!");
            //Write a second line of text
            sw.WriteLine("From the StreamWriter class");
```

```
            //Close the file
            sw.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("Exception: " + e.Message);
        }
        finally
        {
            Console.WriteLine("Executing finally block.");
        }
    }
  }
}
```

Przykład kodu do czytania pliku, wersja 2:

```
//Write a text file - Version 2
using System;
using System.IO;
using System.Text;
namespace readwriteapp
{
   class Class1
   {
      [STAThread]
      static void Main(string[] args)
      {
         Int64 x;
         try
         {
            //Open the File
            StreamWriter sw = new StreamWriter("C:\\Test1.txt", true, Encoding.ASCII);
            //Writeout the numbers 1 to 10 on the same line.
            for (x = 0; x < 10; x++)
            {
               sw.Write(x);
            }
            //close the file
            sw.Close();
         }
         catch (Exception e)
         {
            Console.WriteLine("Exception: " + e.Message);
         }
         finally
         {
            Console.WriteLine("Executing finally block.");
```

```
        }
      }
   }
}
```

Przykład kodu do zapisu w plik tekstowy wersja 2:

```
//Write a text file - Version 2
using System;
using System.IO;
using System.Text;
namespace readwriteapp
{
   class Class1
   {
      [STAThread]
      static void Main(string[] args)
      {
         Int64 x;
         try
         {
            //Open the File
            StreamWriter sw = new StreamWriter("C:\\Test1.txt", true, Encoding.ASCII);
            //Writeout the numbers 1 to 10 on the same line.
            for (x = 0; x < 10; x++)
            {
               sw.Write(x);
            }
            //close the file
            sw.Close();
         }
         catch (Exception e)
         {
            Console.WriteLine("Exception: " + e.Message);
         }
         finally
         {
            Console.WriteLine("Executing finally block.");
         }
      }
   }
}
```

Przykład kodu przetwarzania liczby na znak, wersja 1:

```
using System;

class Program
{
   static void Main()
```

```
    {
        int intValue = 65; // Represents 'A' in ASCII
        char charValue = (char)intValue;

        Console.WriteLine("Convert From Int To Char In C# Using Type Casting");
        Console.WriteLine("Integer Output : " + intValue);
        Console.WriteLine("Character Output : " + charValue);
    }
}
```

Przykład kodu przetwarzania liczby na znak, wersja 2:

```
class Program
{
    static void Main()
    {
        int intValue = 65; // Represents 'A' in ASCII
        char charValue = Convert.ToChar(intValue);

        Console.WriteLine("Convert From Int To Char In C# Using Convert.ToChar()");
        Console.WriteLine("Integer Output : " + intValue);
        Console.WriteLine("Character Output : " + charValue);
    }
}
```