```
void TestOf_CopyString(void)
{
   char acTestSource[] = "Test of copy";
   char acTestDestinationOk[13]="abcdefghijkm";
   char acTestDestinationTooShort[]="abcd";
   enum CompResult eCompResult;
   CopyString(acTestSource, acTestDestinationOk );

   printf("CopyString\n\n");

   printf("Test 1- \n");
   //Test 1 sprawdza, czy funkcja dobrze kopiuje stringi do niepustej tabeli o dobrym rozmiarze
   eCompResult = eCompareString( acTestSource, acTestDestinationOk );
   if ( eCompResult == DIFFERENT )
   {
      printf("ERROR\n");
   }
   else
   {
      printf("OK\n");
   }

   printf("Test 2- \n");
   //Test 2 sprawdza, czy funkcja dobrze kopiuje stringi do niepustej, za krotkiej tabeli
   eCompResult = eCompareString( acTestSource, acTestDestinationTooShort );
   if ( eCompResult == EQUAL )
   {
      printf("ERROR\n");
   }
   else
   {
      printf("OK\n");
   }
}
```

```
void TestOf_eCompareString(void)
{
   char acTest[]="Test 1";
   char acTestCorrect[]="Test 1";
   char acTestIncorrect[]="Test 2";
   char acTestTooShort[]="Test";
   enum CompResult eCompResult;

   printf("\n\neCompareString\n\n");

   printf("Test 1- \n");
   //test 1 sprawdza czy funkcja poprawnie porownuje takie same stringi
   eCompResult = eCompareString(acTest, acTestCorrect);
   if ( eCompResult == EQUAL )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }

   printf("Test 2- \n");
   //test 2 sprawdza czy funkcja poprawnie porownuje rozne stringi
   eCompResult = eCompareString(acTest, acTestIncorrect);
   if ( eCompResult == DIFFERENT )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }

   printf("Test 3- \n");
   //test 3 sprawdza czy funkcja poprawnie porownuje stringi roznej dlugosci
   eCompResult = eCompareString(acTest, acTestTooShort);
   if ( eCompResult == DIFFERENT )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }
}
```

```
void TestOf_AppendString(void)
{
   enum CompResult eCompResult;
   char acTestResultDidntAppend[]="Test";
   char acTestResultCorrect[]="Test 1";
   char acTestSource[]=" 1";
   char acTestDestination[]="Test";

   printf("\n\nAppendString\n\n");

   printf("Test 1- \n");
   //test 1 sprawdza czy funkja przedluza stringa
   AppendString(acTestSource, acTestDestination);
   eCompResult=eCompareString(acTestDestination,acTestResultDidntAppend);
   if ( eCompResult == EQUAL )
   {
      printf("Error\n");
   }
   else
   {
      printf("Ok\n");
   }

   printf("Test 2- \n");
   //test 2 sprawdza czy funkcja poprawnie przedluza stringa
   eCompResult=eCompareString(acTestDestination,acTestResultCorrect);
   if ( eCompResult == DIFFERENT )
   {
      printf("Error\n");
   }
   else
   {
      printf("Ok\n");
   }
}
```

```
void TestOf_ReplaceCharactersInString(void)
{
   char acTestSource[]="test 1";
   char acTestOldChar='t';
   char acTestNewChar='a';
   char acTestResultCorrect[]="aesa 1";
   char acTestResultIncorrect[]="test 1";
   enum CompResult eCompResult;
   ReplaceCharactersInString(acTestSource, acTestOldChar, acTestNewChar);

   printf("\n\nReplaceCharactersInString\n\n");

   printf("Test 1- \n");
   //test 1 sprawdza czy zmienia znaki
   eCompResult = eCompareString(acTestSource, acTestResultIncorrect);
   if ( eCompResult == EQUAL )
   {
     printf("Error\n");
   }
   else
   {
     printf("Ok\n");
   }

   printf("Test 2- \n");
   //test 2 sprawdza poprawne zmienianie znakow
   eCompResult = eCompareString(acTestSource, acTestResultCorrect);
   if ( eCompResult == DIFFERENT )
   {
     printf("Error\n");
   }
   else
   {
     printf("Ok\n");
   }
}
```

```
void TestOf_UIntToHexStr()
{
   char acTestDestinationTooShort[2];
   char acTestDestinationNotEmpty[]="abcdef";
   char acTestDestinationNotEmptyTooLong[9]="abcdefgh";
   unsigned int uiTestSource = 0x12AC;
   unsigned int uiTestSourceShort = 0x1;
   char acTestCorrecResult[]="0x12AC";
   char acTestCorrecResultShort[]="0x0001";
   enum CompResult eCompResult;

   printf("\n\nUIntToHexStr\n\n");

   printf("Test 1-\n");
   //test 1 sprawdza poprawna konwersje na stringa w idealnych warunkach
   UIntToHexStr(uiTestSource, acTestDestinationNotEmpty);
   eCompResult = eCompareString(acTestDestinationNotEmpty, acTestCorrecResult );
   if ( eCompResult == EQUAL )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }

   printf("Test 2-\n");
   //test 2 sprawdza poprawna konwersje na stringa dla za krotkiej tablicy
   UIntToHexStr(uiTestSource, acTestDestinationTooShort);
   eCompResult = eCompareString(acTestDestinationTooShort, acTestCorrecResult );
   if ( eCompResult == EQUAL )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }
```

```
  printf("Test 3-\n");
  //test 3 sprawdza poprawna konwersje na stringa dla za dlugiej tablicy
  UIntToHexStr(uiTestSource, acTestDestinationNotEmptyTooLong);
  eCompResult = eCompareString(acTestDestinationNotEmptyTooLong, acTestCorrecResult );
  if ( eCompResult == EQUAL )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 4-\n");
  //test 4 sprawdza poprawna konwersje na stringa dla malej liczby
  UIntToHexStr(uiTestSourceShort, acTestDestinationNotEmpty);
  eCompResult = eCompareString(acTestDestinationNotEmpty, acTestCorrecResultShort );
  if ( eCompResult == EQUAL )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
}
```

```
void TestOf_eHexStringToUInt()
{
   char acTestCorrect[]="0x12AC";
   char acTestShortCorrect[]="0x1";
   char acTestStartsWith1[]="1x12AC";
   char acTestSecondIsB[]="0b12AC";
   char acTestTooShort[]="0x";
   char acTestTooLong[]="0x121AC";
   unsigned int uiTestResult;
   enum Result eResult;

   printf("\n\neHexStringToUInt\n\n");

   printf("Test 1-\n");
   //test 1 sprawdza eHexStringToUInt dla idealnego stringa
   eResult = eHexStringToUInt( acTestCorrect, &uiTestResult );
   if ( eResult == OK && uiTestResult == 0x12AC )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }

   printf("Test 2-\n");
   //test 2 sprawdza eHexStringToUInt dla najkrotszego poprawnego stringa
   eResult = eHexStringToUInt( acTestShortCorrect, &uiTestResult );
   if ( eResult == OK && uiTestResult == 0x1 )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }
```

```
  printf("Test 3-\n");
  //test 3 sprawdza eHexStringToUInt dla stringa nie zaczynajacego sie od 0
  eResult = eHexStringToUInt( acTestStartsWith1, &uiTestResult );
  if ( eResult == ERROR )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 4-\n");
  //test 4 sprawdza eHexStringToUInt dla stringa nie majacego x jako drugi znak
  eResult = eHexStringToUInt( acTestSecondIsB, &uiTestResult );
  if ( eResult == ERROR )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 5-\n");
  //test 5 sprawdza eHexStringToUInt dla za krotkiego stringa
  eResult = eHexStringToUInt( acTestTooShort, &uiTestResult );
  if ( eResult == ERROR )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
```

```
  printf("Test 6-\n");
  //test 6 sprawdza eHexStringToUInt dla za dlugiego stringa
  eResult = eHexStringToUInt( acTestTooLong, &uiTestResult );
  if ( eResult == ERROR )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

}

void TestOf_AppendUIntToString()
{
  char acTestDestination[]="abcd";
  unsigned int uiTestSource = 0x12AC;
  char acTestResult[]="abcd0x12AC";
  enum CompResult eCompResult;

  printf("\n\nAppendUIntToString\n\n");

  printf("Test 1-\n");
  //Test 1 sprawdza czy poprawnie wydluzono stringa
  AppendUIntToString( uiTestSource, acTestDestination );
  eCompResult = eCompareString(acTestDestination, acTestResult );
  if ( eCompResult == EQUAL )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
}
```

```
void TestOf_ucFindTokensInString()
{
   char acSource1Token[]="Token1";
   char acSource3Tokens[]="Token1 token2 token3";
   char acSource4Tokens[]="Token1 token2 token3 token4";
   char acSourceEmptyString[]="   ";
   char acFirstDelimiter[]="  Token1 token2";
   char ac3DelimitersBetweenTokens[]="Token1   Token2";
   unsigned char ucNumberOfTokens;
   enum CompResult eCompResult;
   printf("\n\nucFindTokensInString\n\n");

   printf("Test 1-\n");
   //test 1 sprawdza czy poprawnie znajduje token
   ucNumberOfTokens = ucFindTokensInString( acSource1Token );
   eCompResult = eCompareString( acSource1Token, asToken[0].uValue.pcString );
   if ( eCompResult == EQUAL && ucNumberOfTokens == 1 )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }

   printf("Test 2-\n");
   //test 2 sprawdza czy poprawnie znajduje kilka tokenow
   ucNumberOfTokens = ucFindTokensInString( acSource3Tokens );
   eCompResult = eCompareString( acSource3Tokens, asToken[0].uValue.pcString );
   if ( eCompResult == EQUAL && ucNumberOfTokens == 3 )
   {
     printf("Ok\n");
   }
   else
   {
     printf("Error\n");
   }
```

```
  printf("Test 3-\n");
  //test 3 sprawdza czy nie wpisuje wiecej niz 3 tokenow
  ucNumberOfTokens = ucFindTokensInString( acSource4Tokens );
  eCompResult = eCompareString( acSource4Tokens, asToken[0].uValue.pcString );
  if ( eCompResult == EQUAL && ucNumberOfTokens == 3 )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 4-\n");
  //test 4 sprawdza czy nie wpisuje pustego stringa jako token
  ucNumberOfTokens = ucFindTokensInString( acSourceEmptyString );
  if ( ucNumberOfTokens == 0 )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 5-\n");
  //test 5 sprawdza czy dziala jak pierwszymi znakami sa delimitery
  ucNumberOfTokens = ucFindTokensInString( acFirstDelimiter );
  eCompResult = eCompareString( acFirstDelimiter+2, asToken[0].uValue.pcString );
  if ( eCompResult == EQUAL && ucNumberOfTokens == 2 )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
```

```
  printf("Test 6-\n");
  //test 6 sprawdza czy dziala jak miedzy tokenami jest wiecej niz jeden delimiter
  ucNumberOfTokens = ucFindTokensInString( ac3DelimitersBetweenTokens );
  eCompResult = eCompareString( ac3DelimitersBetweenTokens, asToken[0].uValue.pcString );
  if ( eCompResult == EQUAL && ucNumberOfTokens == 2 )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

}

void TestOf_eStringToKeyword()
{
  char acCorrectReset[]="reset";
  char acCorrectLoad[]="load";
  char acCorrectStore[]="store";
  char acIncorrectKeyword[]="lOad";
  char acKeywordWithDelimiter[]=" reset ";
  enum Result eResult;
  enum KeywordCode eKeyCode;
  printf("\n\neStringToKeyword\n\n");

  printf("Test 1-\n");
  //test 1 sprawdza czy dobrze wpisuje keyword reset
  eResult = eStringToKeyword(acCorrectReset, &eKeyCode );
  if ( eResult == OK && eKeyCode == RST )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
```

```
  printf("Test 2-\n");
  //test 2 sprawdza czy dobrze wpisuje keyword load
  eResult = eStringToKeyword(acCorrectLoad, &eKeyCode );
  if ( eResult == OK && eKeyCode == LD )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 3-\n");
  //test 3 sprawdza czy dobrze wpisuje keyword store
  eResult = eStringToKeyword(acCorrectStore, &eKeyCode );
  if ( eResult == OK && eKeyCode == ST )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }

  printf("Test 4-\n");
  //test 4 sprawdza czy dobrze odnajduje keyword
  eResult = eStringToKeyword(acIncorrectKeyword, &eKeyCode );
  if ( eResult == ERROR )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
```

```
  printf("Test 5-\n");
  //test 5 sprawdza czy dziala z delimiterami
  eResult = eStringToKeyword(acKeywordWithDelimiter, &eKeyCode );
  if ( eResult == ERROR )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
}

void TestOf_DecodeTokens()
{
  char acCorrectKeywordNumberString[]="reset 0x12AC  string";
  char acIncorrectNumbers[]="1x12AC 0b12AC 0x";

  printf("\n\nDecodeTokens\n\n");

  printf("Test 1-\n");
  //test 1 sprawdza czy poprawnie dekoduje keyword number i string
  ucFindTokensInString( acCorrectKeywordNumberString );
  ReplaceCharactersInString(acCorrectKeywordNumberString, ' ', NULL);
  DecodeTokens();
  if ( asToken[0].uValue.eKeyword == RST && asToken[0].eType == KEYWORD && asToken[1].uValue.uiNumber == 0x12AC &&
asToken[1].eType == NUMBER  && asToken[2].eType == STRING )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
```

```
  printf("Test 2-\n");
  //test 2 sprawdza czy liczby w zlym formacie traktuje jako string
  ucFindTokensInString( acIncorrectNumbers );
  ReplaceCharactersInString(acIncorrectNumbers, ' ', NULL);
  DecodeTokens();
  if ( asToken[0].eType == STRING && asToken[1].eType == STRING  && asToken[2].eType == STRING )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
}

void TestOf_DecodeMsg()
{
  char acTestMessage[]="store 0x12AC quickly";

  printf("\n\nDecodeMsg\n\n");

  printf("Test 1-\n");
  //test 1 sprawdza czy poprawnie dekoduje caly komunikat
  DecodeMsg(acTestMessage);
  if ( asToken[0].eType == KEYWORD && asToken[0].uValue.eKeyword==ST && asToken[1].eType == NUMBER && asToken[1].uValue.uiNumber
== 0x12AC && asToken[2].eType == STRING )
  {
    printf("Ok\n");
  }
  else
  {
    printf("Error\n");
  }
}
```