```
//*****************************************************************************
//*****************************************************************************
//                                                    **
//                        Testy          **
//*****************************************************************************

#include <stdio.h>
#include "./dekodowanie.h"
#include "./konwersje.h"
#include "./operacje_proste.h"

#define NULL '\0'
#define NOTEQUAL DIFFERENT

void TestOf_CopyString(void){
   char cSource1[] = "abc";
   char cDest1[10];
   char cSource2[] = "";
   char cDest2[10];

   printf("bCopyString\n\n ");

   printf ("Test 1 - ");
   // skopiowanie prostego łańcucha znakowego
   CopyString(cSource1, cDest1);
   if (EQUAL == eCompareString(cSource1, cDest1)) printf("OK\n"); else printf("Error\n");

   printf ("Test 2 - ");
   // skopiowanie pustego łańcucha
   CopyString(cSource2, cDest2);
   if (EQUAL == eCompareString(cSource2, cDest2)) printf("OK\n"); else printf("Error\n");
}
```

```
void TestOf_eCompareString(void){
   char cStr1_1[] = "abc";
   char cStr1_2[] = "abc";
   char cStr2_1[] = "abc";
   char cStr2_2[] = "abd";
   char cStr3_1[] = "";
   char cStr3_2[] = "";
   char cStr4_1[] = "abc";
   char cStr4_2[] = "";

   printf("bCompareString\n\n ");

   printf ("Test 1 - ");
     // porównanie dwóch identycznych łańcuchów znaków
   if (EQUAL == eCompareString(cStr1_1, cStr1_2)) printf("OK\n"); else printf("Error\n");

   printf ("Test 2 - ");
     // porównanie dwóch różnych łańcuchów znaków
   if (NOTEQUAL == eCompareString(cStr2_1, cStr2_2)) printf("OK\n"); else printf("Error\n");

   printf ("Test 3 - ");
     // porównanie pustych łańcuchów
   if (EQUAL == eCompareString(cStr3_1, cStr3_2)) printf("OK\n"); else printf("Error\n");

   printf ("Test 4 - ");
     // porównanie łańcucha z pustym łańcuchem
   if (NOTEQUAL == eCompareString(cStr4_1, cStr4_2)) printf("OK\n"); else printf("Error\n");
}
```

```
void TestOf_AppendString(void){
   char cDest1[20] = "Hello";
   char cSrc1[] = "World";
   char cExpected1[] = "HelloWorld";
   char cDest2[20] = "Data";
   char cSrc2[] = "";
   char cExpected2[] = "Data";
   char cDest3[20] = "";
   char cSrc3[] = "Start";
   char cExpected3[] = "Start";
   char cDest4[20] = "";
   char cSrc4[] = "";
   char cExpected4[] = "";

   printf("bAppendString\n\n ");

   printf ("Test 1 - ");
   // dodanie dwóch niepustych łańcuchów
   AppendString(cSrc1, cDest1);
   if (EQUAL == eCompareString(cDest1, cExpected1)) printf("OK\n"); else printf("Error\n");

   printf ("Test 2 - ");
   // dodanie pustego źródła do niepustego celu
   AppendString(cSrc2, cDest2);
   if (EQUAL == eCompareString(cDest2, cExpected2)) printf("OK\n"); else printf("Error\n");

   printf ("Test 3 - ");
   // dodanie niepustego źródła do pustego celu
   AppendString(cSrc3, cDest3);
   if (EQUAL == eCompareString(cDest3, cExpected3)) printf("OK\n"); else printf("Error\n");

   printf ("Test 4 - ");
   // dodanie pustego źródła do pustego celu
   AppendString(cSrc4, cDest4);
   if (EQUAL == eCompareString(cDest4, cExpected4)) printf("OK\n"); else printf("Error\n");
}
```

```
void TestOf_ReplaceCharactersInString(void){
   char cStr1[] = "banana";
   char cExpected1[] = "bonono";
   char cStr2[] = "apple";
   char cExpected2[] = "apple";

   printf("bReplaceCharactersInString\n\n ");

   printf ("Test 1 - ");
   // zamiana jednego znaku występującego wielokrotnie
   ReplaceCharactersInString(cStr1, 'a', 'o');
   if (EQUAL == eCompareString(cStr1, cExpected1)) printf("OK\n"); else printf("Error\n");

   printf ("Test 2 - ");
   // zamiana znaku, który nie występuje w łańcuchu
   ReplaceCharactersInString(cStr2, 'z', 'x');
   if (EQUAL == eCompareString(cStr2, cExpected2)) printf("OK\n"); else printf("Error\n");
}


void TestOf_UIntToHexStr(void){
   char cStr1[7];
   char cExpected1[] = "0x0000";
   char cStr2[7];
   char cExpected2[] = "0x00FF";

   printf("bUIntToHexStr\n\n ");

   printf ("Test 1 - ");
   // konwersja liczby 0x0000
   UIntToHexStr(0x0000, cStr1);
   if (EQUAL == eCompareString(cStr1, cExpected1)) printf("OK\n"); else printf("Error\n");

   printf ("Test 2 - ");
   // konwersja liczby 0x00FF
   UIntToHexStr(0x00FF, cStr2);
   if (EQUAL == eCompareString(cStr2, cExpected2)) printf("OK\n"); else printf("Error\n");
}
```

```
void TestOf_eHexStringToUInt(void){
   char cStr1[] = "0x1A3F";
   unsigned int uiExpected1 = 0x1A3F;
   unsigned int uiResult1;
   char cStr2[] = "1A3F";
   unsigned int uiResult2;
   char cStr3[] = "0x1AGF";
   unsigned int uiResult3;
   char cStr4[] = "0x";
   unsigned int uiResult4;
   printf("bHexStringToUInt\n\n ");

   printf ("Test 1 - ");
   // poprawny ciąg hex z cyframi i literami
   if (OK == eHexStringToUInt(cStr1, &uiResult1) && uiResult1 == uiExpected1) printf("OK\n"); else printf("Error\n");

   printf ("Test 2 - ");
   // niepoprawny prefiks (brak 0x)
   if (ERROR == eHexStringToUInt(cStr2, &uiResult2)) printf("OK\n"); else printf("Error\n");

   printf ("Test 3 - ");
   // zawiera niepoprawny znak (litera spoza A-F)
   if (ERROR == eHexStringToUInt(cStr3, &uiResult3)) printf("OK\n"); else printf("Error\n");

   printf ("Test 4 - ");
   // pusty ciąg po prefiksie
   if (ERROR == eHexStringToUInt(cStr4, &uiResult4)) printf("OK\n"); else printf("Error\n");
}

void TestOf_AppendUIntToString(void){
  char cDest1[20] = "";
  char cExpected1[] = "0x1A3F";
  char cDest2[30] = "Value: ";
  char cExpected2[] = "Value: 0x2B";
  printf("bAppendUIntToString\n\n ");

  printf ("Test 1 - ");
  // dodanie liczby hex do pustego łańcucha
  AppendUIntToString(0x1A3F, cDest1);
  if (EQUAL == eCompareString(cDest1, cExpected1)) printf("OK\n"); else printf("Error\n");

  printf ("Test 2 - ");
  // dodanie liczby hex do niepustego łańcucha
  AppendUIntToString(0x2B, cDest2);
  if (EQUAL == eCompareString(cDest2, cExpected2)) printf("OK\n"); else printf("Error\n");
}
```

```
void TestOf_ucFindTokensInString(void) {
   char cString1[] = "test of this";
   char cString2[] = "     ";
   char cString3[] = " test of this";
   char cString4[] = "test  of this";

   printf("ucFindTokensInString\n\n ");

   printf("Test 1 - ");
   // sprawdzenie poprawności dzielenia typowego łańcucha na tokeny
   ucTokenNr = ucFindTokensInString(cString1);
   if (ucTokenNr == 3 &&
     EQUAL == eCompareString(asToken[0].uValue.pcString, "test of this") &&
     EQUAL == eCompareString(asToken[1].uValue.pcString, "of this") &&
     EQUAL == eCompareString(asToken[2].uValue.pcString, "this")) printf("OK\n"); else printf("Error\n");

   printf("Test 2 - ");
   // sprawdzenie działania dla łańcucha składającego się wyłącznie ze spacji
   ucTokenNr = ucFindTokensInString(cString2);
   if (ucTokenNr == 0) printf("OK\n"); else printf("Error\n");

   printf("Test 3 - ");
   // sprawdzenie działania dla łańcucha ze spacją na początku
   ucTokenNr = ucFindTokensInString(cString3);
   if (ucTokenNr == 3 &&
     EQUAL == eCompareString(asToken[0].uValue.pcString, "test of this") &&
     EQUAL == eCompareString(asToken[1].uValue.pcString, "of this") &&
     EQUAL == eCompareString(asToken[2].uValue.pcString, "this")) printf("OK\n"); else printf("Error\n");

   printf("Test 4 - ");
   // sprawdzenie działania dla podwójnej spacji między słowami
   ucTokenNr = ucFindTokensInString(cString4);
   if (ucTokenNr == 3 &&
     EQUAL == eCompareString(asToken[0].uValue.pcString, "test  of this") &&
     EQUAL == eCompareString(asToken[1].uValue.pcString, "of this") &&
     EQUAL == eCompareString(asToken[2].uValue.pcString, "this")) printf("OK\n\n"); else printf("Error\n\n");
}
```

```
void TestOf_eStringToKeyword(void) {
   char cString1[] = "load";
   char cString2[] = "loooaddd";
   enum KeywordCode eTest;

   printf("eStringToKeyword\n\n ");

   printf("Test 1 - ");
   // sprawdzenie poprawnej konwersji słowa kluczowego na wartość typu enum
   if ((eStringToKeyword(cString1, &eTest) == OK) && eTest == LD) printf("OK\n"); else printf("Error\n");

   printf("Test 2 - ");
   // sprawdzenie działania funkcji dla niepoprawnego słowa
   if (eStringToKeyword(cString2, &eTest) == ERROR) printf("OK\n\n"); else printf("Error\n\n");
}


void TestOf_DecodeTokens(void) {
   char cString[] = "0x10 reset test";
   ucTokenNr = ucFindTokensInString(cString);
   ReplaceCharactersInString(cString, DELIMITER_CHARACTER, NULL);
   DecodeTokens();

   printf("DecodeTokens\n\n ");

   printf("Test 1 - ");
   // sprawdzenie poprawnej interpretacji tokena będącego zwykłym łańcuchem znaków
   if ((asToken[2].eType == STRING) && (asToken[2].uValue.pcString == &cString[11])) printf("OK\n"); else printf("Error\n");

   printf("Test 2 - ");
   // sprawdzenie poprawnej interpretacji tokena będącego słowem kluczowym
   if ((asToken[1].eType == KEYWORD) && (asToken[1].uValue.eKeyword == RST)) printf("OK\n"); else printf("Error\n");

   printf("Test 3 - ");
   // sprawdzenie poprawnej interpretacji tokena będącego liczbą w formacie hex
   if ((asToken[0].eType == NUMBER) && (asToken[0].uValue.uiValue == 0x10)) printf("OK\n\n"); else printf("Error\n\n");
}
```

```
void TestOf_DecodeMsg(void) {
   char cStr[] = "test reset 0x10";

   printf("DecodeMsg\n\n ");

   printf("Test 1 - ");
   // sprawdzenie poprawnej interpretacji pełnej wiadomości zawierającej tekst, słowo kluczowe i liczbę
   DecodeMsg(cStr);
   if ((asToken[0].eType == STRING) && (EQUAL == eCompareString("test", asToken[0].uValue.pcString)) &&
      (asToken[1].eType == KEYWORD) && (asToken[1].uValue.eKeyword == RST) &&
      (asToken[2].eType == NUMBER) && (asToken[2].uValue.uiValue == 0x10)) printf("OK\n\n"); else printf("Error\n\n");
}


int main() {
  printf("TESTY FUNKCJI DO LANCUCHY ZNAKOWE\n\n ");
  TestOf_CopyString();
  TestOf_eCompareString();
  TestOf_AppendString();
  TestOf_ReplaceCharactersInString();

  printf("TESTY FUNKCJI DO KONWERSJI\n\n ");
  TestOf_UIntToHexStr();
  TestOf_eHexStringToUInt();
  TestOf_AppendUIntToString();

  printf("TESTY FUNKCJI DO DEKODOWANIA\n\n ");
  TestOf_ucFindTokensInString();
  TestOf_eStringToKeyword();
  TestOf_DecodeTokens();
  TestOf_DecodeMsg();
}
```