void TestOf\_CopyString(void) {

printf("CopyString\n\n ");

printf("Test 1 - ");

//o takiej samej dlugosci

char cCopyVersion1[]="tyuiop";

CopyString("qwerty", cCopyVersion1);

if (strcmp("qwerty", cCopyVersion1)==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//pierwszy jest krotszy

char cCopyVersion2[]="qwerty";

CopyString("rty", cCopyVersion2);

if (strcmp("rty", cCopyVersion2)==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 3 - ");

//pierwszy jest pusty

char cAfterCopy3[]="qwerty";

CopyString("", cAfterCopy3);

if (strcmp("", cAfterCopy3)==0) printf("OK\n\n "); else printf("Error\n\n ");

}

void TestOf\_eCompareString(void) {

printf("eCompareString\n\n ");

printf("Test 1 - ");

//takie same, o tej samej dlugosci

if (eCompareString("qwerty", "qwerty")==EQUAL) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//rozne, ale o tej samej dlugosci

if (eCompareString("qwerty", "tyuiop")==DIFFERENT) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 3 - ");

///pierwszy jest pusty

if (eCompareString("", "uiop")==DIFFERENT) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 4 - ");

//drugi jest pusty

if (eCompareString("qwerty", "")==DIFFERENT) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 5 - ");

///oba sa puste

if (eCompareString("", "")==EQUAL) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 6 - ");

///drugi jest dluzszy

if (eCompareString("qwerty", "qwertyu")==DIFFERENT) printf("OK\n\n\n"); else printf("Error\n\n\n");

}

void TestOf\_AppendString(void) {

printf("AppendString\n\n ");

printf("Test 1 - ");

//dowolna zawartosc

char cAppendStrTest1[]="tyuiop";

AppendString("qwerty", cAppendStrTest1);

if (strcmp(cAppendStrTest1, "tyuiopqwerty")==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//pierwszy jest pusty

char cAppendStrTest2[]="qwerty";

AppendString("", cAppendStrTest2);

if (strcmp(cAppendStrTest2, "qwerty")==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 3 - ");

//drugi pusty

char cAppendStrTest3[]="";

AppendString("qwerty", cAppendStrTest3);

if (strcmp(cAppendStrTest3, "qwerty")==0) printf("OK\n\n\n"); else printf("Error\n\n\n");

}

void TestOf\_ReplaceCharactersInString(void) {

printf("ReplaceCharactersInString\n\n ");

printf("Test 1 - ");

//zamiana znakow

char cOrginalStr1[]="ala ma kota";

ReplaceCharactersInString(cOrginalStr1, 'a', 'A');

if (strcmp(cOrginalStr1, "AlA mA kotA")==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//zmiana ze spacji na NULL

char cOrginalStr2[]="ala ma kota";

ReplaceCharactersInString(cOrginalStr2, ' ', '\0');

if (strcmp(cOrginalStr2, "ala\0ma\0kota")==0) printf("OK\n\n"); else printf("Error\n\n");

}

void TestOf\_UIntToHexStr(void) {

printf("UIntToHexStr\n\n ");

printf("Test 1 - ");

//konwersja na lancuch tekstowy

char cAfterChange[6];

UIntToHexStr(0x079A, cAfterChange);

if (strcmp(cAfterChange, "0x079A")==0) printf("OK\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//zera po 0x w lancuchu

UIntToHexStr(0x21, cAfterChange);

if (strcmp(cAfterChange, "0x0021")==0) printf("OK\n "); else printf("Error\n\n");

}

void TestOf\_eHexStringToUInt() {

enum Result eReturnResult;

unsigned int uiReturnValue;

printf("eHexStringToUInt\n\n ");

printf("Test 1 - ");

//po tetradach 0,7,9,A

eReturnResult = eHexStringToUInt("0x079A", &uiReturnValue);

if (eReturnResult==OK) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//za krotki

eReturnResult = eHexStringToUInt("0x", &uiReturnValue);

if (eReturnResult==ERROR) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 3 - ");

//za dlugi

eReturnResult = eHexStringToUInt("0xEEAB2", &uiReturnValue);

if (eReturnResult==ERROR) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 4 - ");

//poczatek rozny od 0x

eReturnResult = eHexStringToUInt("qwerty", &uiReturnValue);

if (eReturnResult==ERROR) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 5 - ");

//mniej znakow

eReturnResult = eHexStringToUInt("0x034", &uiReturnValue);

if (eReturnResult==OK) printf("OK\n\n"); else printf("Error\n\n");

}

void TestOf\_AppendUIntToString() {

printf("AppendUIntToString\n\n ");

printf("Test 1 - ");

//dopisywanie od NULLa stringa

char cOrginal1[]="qwerty uiop";

AppendUIntToString(0xAB12,cOrginal1);

if (strcmp(cOrginal1,"qwerty uiop0xAB12")==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//pusty string

char cOrginal2[]="";

AppendUIntToString(0xAB12,cOrginal2);

if (strcmp(cOrginal2,"0xAB12")==0) printf("OK\n\n\n"); else printf("Error\n\n\n");

}

void TestOf\_ucFindTokensInString() {

unsigned char ucTokenNumber;

printf("ucFindTokensInString\n\n ");

printf("Test 1 - ");

//maksymalna liczba tokenow

char pcTokens1[]="0x31AB QWERTY reset";

ucTokenNumber=ucFindTokensInString(pcTokens1);

if ((ucTokenNumber==3)&&(&pcTokens1[0]==asToken[0].uValue.pcString)&&(&pcTokens1[7]==asToken[1].uValue.pcString) &&

(&pcTokens1[14]==asToken[2].uValue.pcString)) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//tylko delimitery

char pcTokens2[]=" ";

ucTokenNumber=ucFindTokensInString(pcTokens2);

if (ucTokenNumber==0) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 3 - ");

//delimiter przed pierwszym tokenem, mniej niz 3 tokeny

char pcTokens3[]=" 0x31AB QWERTY ";

ucTokenNumber=ucFindTokensInString(pcTokens3);

if ((ucTokenNumber==2)&&(&pcTokens3[3]==asToken[0].uValue.pcString)&&

(&pcTokens3[10]==asToken[1].uValue.pcString)) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 4 - ");

//pomiedzy tokenami wiecej niz jeden delimiter

char pcTokens4[]="0x31AB QWERTY reset";

ucTokenNumber=ucFindTokensInString(pcTokens4);

if ((ucTokenNumber==3)&&(&pcTokens4[0]==asToken[0].uValue.pcString)&&

(&pcTokens4[9]==asToken[1].uValue.pcString) && (&pcTokens4[21]==asToken[2].uValue.pcString)) printf("OK\n\n "); else printf("Error\n\n ");

}

void TestOf\_eStringToKeyword() {

enum KeywordCode eTestCode;

printf("eStringToKeyword\n\n ");

printf("Test 1 - ");

//reset - slowo kluczowe

if (eStringToKeyword("reset", &eTestCode)==OK) printf("OK\n\n "); else printf("Error\n\n ");

printf("Test 2 - ");

//slowo inne niz kluczowe

if (eStringToKeyword("qwerty", &eTestCode)==ERROR) printf("OK\n\n\n"); else printf("Error\n\n\n");

printf("Test 3 - ");

//sprawdzenie przekodowania

eStringToKeyword("store", &eTestCode);

if (eTestCode == asKeywordList[2].eCode) printf("OK\n"); else printf("ERROR\n");

}

void TestOf\_DecodeTokens() {

unsigned char ucTokenNumber;

printf("DecodeTokens\n\n ");

printf("Test 1 - ");

//liczba i zapis jej, string i ustawienie wskaznika, keyword i zapis kodu

char pcTokens[]="0x31AB\0QWERTY\0 reset";

asToken[0].uValue.pcString = &pcTokens[0];

asToken[1].uValue.pcString = &pcTokens[7];

asToken[2].uValue.pcString = &pcTokens[15];

DecodeTokens();

if ((asToken[0].eType==NUMBER) && (asToken[0].uValue.uiNumber==0x31AB) &&

(asToken[1].eType==STRING) && (&pcTokens[7]==asToken[1].uValue.pcString) &&

(asToken[2].eType==KEYWORD) && (asToken[2].uValue.eKeyword==RST)) printf("OK\n\n\n"); else printf("Error\n\n\n");

}

void TestOf\_DecodeMsg() {

char pcDecodeTokens[]="0x31AB QWERTY reset";

printf("DecodeMsg\n\n ");

printf("Test 1 - ");

//liczba, string, keyword

DecodeMsg(pcDecodeTokens);

if ((asToken[0].eType==NUMBER) && (asToken[0].uValue.uiNumber==0x31AB) &&

(asToken[1].eType==STRING) && (&pcDecodeTokens[7]==asToken[1].uValue.pcString) &&

(asToken[2].eType==KEYWORD) && (asToken[2].uValue.eKeyword==RST)) printf("OK\n\n\n"); else printf("Error\n\n\n");

}

int main(void) {

printf("TESTY FUNKCJI - LANCUCHY ZNAKOWE - OPERACJE PROSTE\n\n\n");

TestOf\_CopyString();

TestOf\_eCompareString();

TestOf\_AppendString();

TestOf\_ReplaceCharactersInString();

printf("TESTY FUNKCJI - LANCUCHY ZNAKOWE - KONWERSJE\n\n\n");

TestOf\_UIntToHexStr();

TestOf\_eHexStringToUInt();

TestOf\_AppendUIntToString();

printf("TESTY FUNKCJI - DEKODOWANIE KOMUNIKATOW\n\n");

TestOf\_ucFindTokensInString();

TestOf\_eStringToKeyword();

TestOf\_DecodeTokens();

TestOf\_DecodeMsg();

}