INTEX | Centrum Szkoleniowe Systemów Automatyki

PODRĘCZNIK

WPROWADZENIE DO JĘZYKA SKRYPTOWEGO C

Ważne wskazówki dotyczące bezpieczeństwa!

Przykładowe aplikacje i rozwiązanie zawarte w tym podręczniku należy traktować jako uproszczone i niekompletne pod względem przedstawionego schematu oraz warunków, jakie należy uwzględnić w rzeczywistej aplikacji. INTEX Sp. z o.o. nie odpowiada za poprawność i kompletność aplikacji tworzonych przez uczestników szkolenia. Ponieważ opisane w podręczniku ćwiczenia w trakcie szkolenia są przeprowadzane z wykorzystaniem dedykowanego stanowiska szkoleniowego, niezależnie od sposobu ich realizacji w żadnym wypadku nie dojdzie do uszkodzenia mienia ani zranienia osób.

Uczestnik szkolenia/użytkownik dokumentacji musi jednak mieć świadomość, że każda ingerencja w system sterowania maszyną/instalacją wiąże się z dużym zagrożeniem!

W wyniku ingerencji, istniejące funkcje bezpieczeństwa mogą zostać wyłączone lub pominięte. Część instalacji może zostać w sposób niezamierzony lub niebezpieczny uruchomiona, zatrzymana, zasilona lub wprawiona w ruch. Zdarzenia te w następstwie mogą doprowadzić do przerwy w produkcji, szkód materialnych czy też niebezpieczeństwa zranienia lub śmierci personelu obsługi.

Każda ingerencja w system sterowania maszyną/instalacją podlega z tego powodu szczególnym wymaganiom bezpieczeństwa i dlatego może być przeprowadzona tylko i wyłącznie pod ścisłym nadzorem doświadczonego i odpowiednio uprawnionego personelu technicznego!

Spis treści

Wprowadzenie do języka C - wersja 160912

I. Wstęp

Celem niniejszego podręcznika jest zapoznanie z językiem C tych osób, które nie miały wcześniej kontaktu z programowaniem.

Język C wykorzystywany w WinCC do tworzenia skryptów rozszerza możliwości pakietu o tworzenie własnych, często bardzo rozbudowanych funkcji.

Przykłady stosowania skryptów:

- realizacja operacji na oknach dialogowych - poprawa wyglądu wizualizacji, oraz czytelności projektowanej aplikacji
- realizacja przeliczeń danych potrzebnych do wyświetlenia w systemie a niedostępnych w sposób bezpośredni
- realizacje operacji na archiwach - zapis i odczyt danych koniecznych do archiwizacji w systemie, operacje na plikach
- realizacje operacji pomocniczych np. identyfikacja użytkowników systemu, uruchamianie innych aplikacji systemu *Windows*
- komunikacja z kontrolkami *ActiveX* poprawiającymi użyteczność aplikacji
- dynamizacja obiektów wizualizacji zależnie od potrzeb systemu
- zapamiętywanie danych w plikach - danych do receptur, danych dotyczących ustawień systemu itd.

Informacje zawarte w podręczniku ograniczone są do minimum potrzebnego dla łatwiejszego zrozumienia szkolenia WIZUALIZACJA organizowanego przez INTEX Sp. z o.o. (szczegóły dotyczące tego szkolenia dostępne są na stronie www.intex.com.pl).

Jeśli czytelnik jest zainteresowany rozszerzeniem swojej wiedzy na temat programowania w języku C, to godnymi polecenia są następujące pozycje książkowe:

- *Brian W. Kernighan, Dennis M. Ritche "Język C"* jedna z najlepszych książek o języku C w jego podstawowej składni *AnsiC*;
- *Jon Bates, Tim Tompkins "Poznaj Visual C++"* książka o środowisku programistycznym Microsoft Visual C++ z której także można dużo dowiedzieć się o podstawach języka C (w oparciu o pakiet Visual C++);

Omawiane w podręczniku przykłady można próbować przećwiczyć samodzielnie w oparciu o jeden z darmowych kompilatorów języka C. Na stronach internetowych można znaleźć kilka darmowych kompilatorów języka C jednym z nich może być kompilator udostępniany przez Borland poprzez strony www.borland.pl.

Podręcznik ten składa się z trzech części:

- **Wprowadzenie do języka C** opisujące podstawowe elementy składni języka C
- **Ćwiczenia praktyczne** to przykłady wykorzystania języka C do realizacji prostych zadań
- **Rozwiązania zadań** zawierające odpowiedzi do pytań testowych umieszczonych w części **Ćwiczenia praktyczne**.

II. Wprowadzenie do języka C

II-1. Struktura

W programie napisanym w języku C można wyróżnić podstawowe elementy:

- dyrektywy preprocesora
- komentarze
- deklaracje
- instrukcje.

Dyrektywy preprocesora

Dyrektywy preprocesora zajmują pełne wiersze i zaczynają się od znaku #.
Dyrektywy te zostaną wykonane jeszcze przed kompilacją programu.

Najczęściej wykorzystywanymi dyrektywami są:

#include <nazwa pliku> dyrektywa oznaczająca dołączenie pliku nagłówkowego do programu przed jego kompilacją. W pliku takim możemy np. umieścić najczęściej używane funkcje własne wykorzystywane później w dalszym programie.

#define <nazwa> <wartość> wszędzie gdzie umieścimy w programie odwołanie do <nazwa> preprocesor umieści <wartość>. Poprawia to czytelność programu – zamiast wpisywać liczby niezrozumiałe dla człowieka możemy wpisywać nazwy symboliczne.

Na końcu linii z dyrektywą preprocesora nie może być znaku **;**

Komentarze

W języku C można wpisywać całe linie komentarza: **//Komentarz**
Można również komentarze „wtrącać" w kontekst programu: **/* Komentarz */**

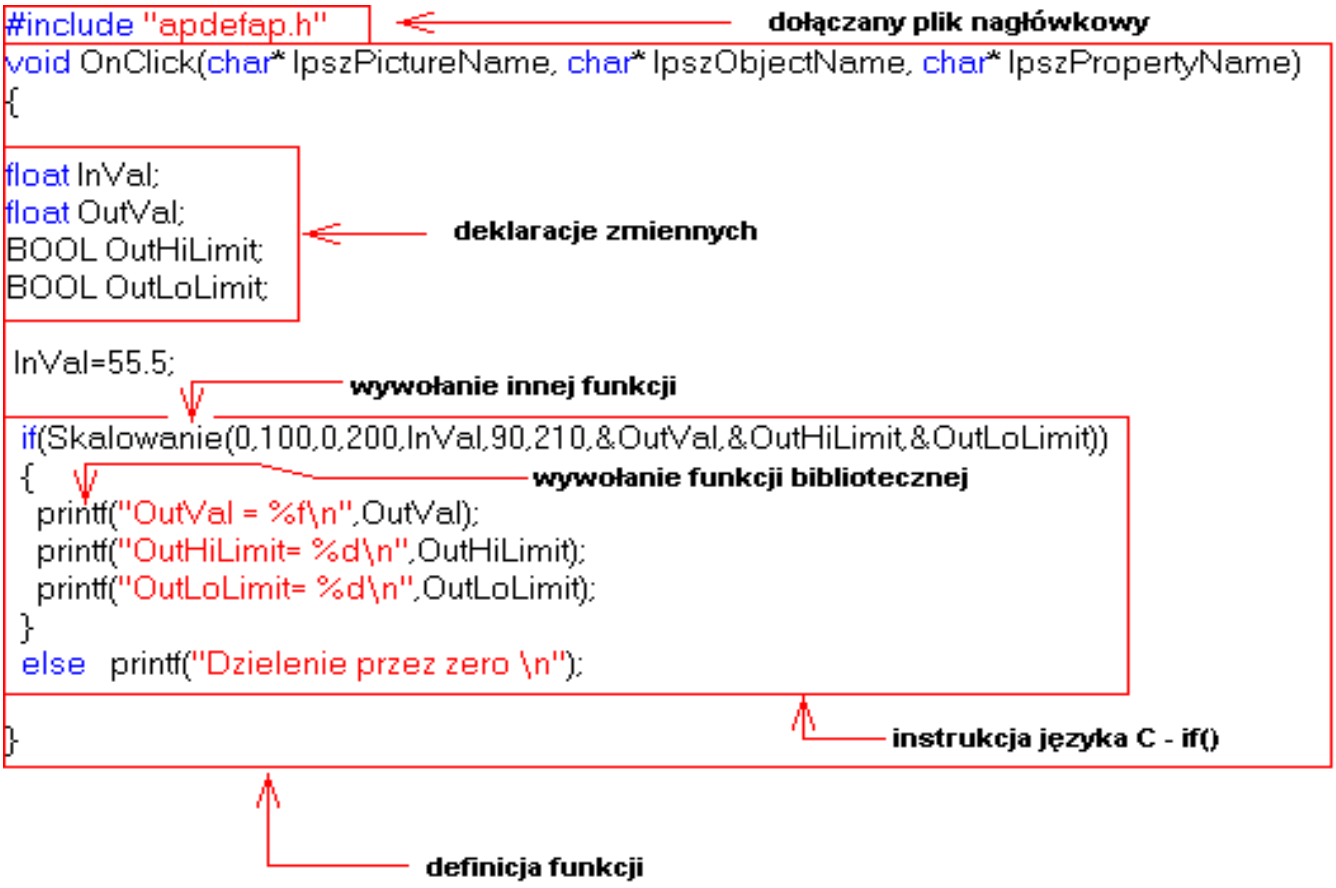
Deklaracje

Tutaj umieszczane są definicje zmiennych, bądź funkcji użytych w programie.

Instrukcje

Kolejne linie programu muszą kończyć się znakiem **;** .
Program realizowany w określonej instrukcji (bądź funkcji) powinien zawierać się między znakami **{** i **}** , które informują kompilator o początku i końcu instrukcji. Jeśli wewnątrz danej instrukcji wykonywana jest tylko jedna linia programu to nie trzeba używać znaków rozdzielających **{** , **}** .

Poniżej przedstawiono przykład prostego programu zapisanego w języku C (skryptu zaczerpniętego z *WinCC*):

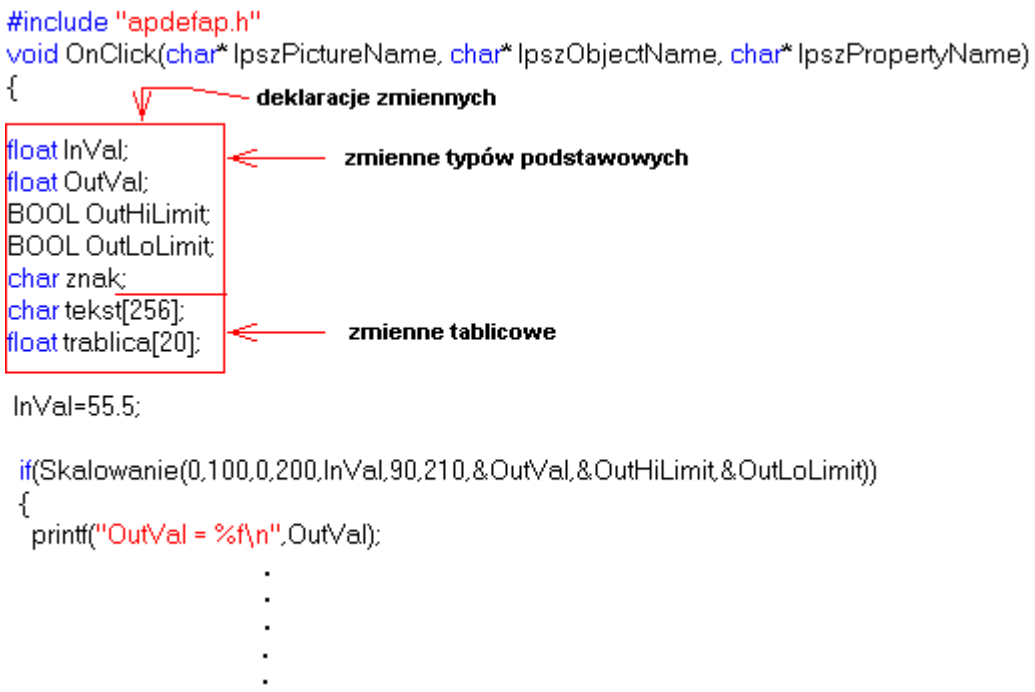


II-2. Zmienne

Zmienna w języku programowania oznacza zarezerwowany obszar w pamięci operacyjnej komputera, który przechowuje dane potrzebne do realizacji określonych operacji np. obliczeń.

Rozmiar rezerwowanej pamięci zależy od typu zmiennej. Programista ma dostęp do danych związanych ze zmienną poprzez jej nazwę określoną w deklaracji zmiennej. W deklaracji zmiennej zawsze należy podać typ zmiennej oraz jej nazwę.

Oto przykład deklaracji zmiennych zaczerpnięty z typowego programu:



W języku C dostępnych jest kilka podstawowych typów zmiennych:

TYP	OPIS
char	typ znakowy, kody znaków ASCII
int	liczby całkowite
float	liczby zmiennoprzecinkowe
short	liczby całkowite krótkie
long int	liczby całkowite długie
double	liczby zmiennoprzecinkowe podwójnej precyzji
long double	liczby zmiennoprzecinkowe podwójnej precyzji długie
unsigned	zmienne "bez znaku"
BOOL	zmienna całkowita, która przyjmuje dwie wartości: TRUE - prawda, FALSE - fałsz
BYTE	pojedynczy bajt
WORD	słowo
DWORD	podwójne słowo

Rozmiar danych przechowywanych przez poszczególne typy nie zawsze jest jednoznaczny

- zależy od typu kompilatora i rodzaju systemu, na który dany kompilator jest przeznaczony.

Przykładowe deklaracje zmiennych:

```
int i;                  //definicja zmiennej i typu int
float zm1=3.66;         //definicja zmiennej zm1 typu float o wartości początkowej '3.66'
char znak='a';          //definicja zmiennej znak typu char o wartości początkowej 'a'
```

Tablice w C są definiowane poprzez podanie typu zmiennych wchodzących w skład tablicy oraz rozmiaru tablicy. Tablica stanowi zestaw wielu zmiennych (ilość zależy od rozmiaru tablicy) tego samego typu.

Do danego elementu w tablicy można uzyskać dostęp poprzez indeks - czyli położenie w tablicy.

Przykładowa deklaracja tablicy składającej się z 1000 elementów typu float:

```
float tablica[1000];
```

Przykładowa deklaracja tablicy składającej się z 255 elementów typu char - tablica znaków (łańcuch tekstowy):

```
char tablica[256];
```

Teksty w języku C są zapisywane w tablicach znakowych w określony sposób. Na początku tablicy tekstów - licząc od zerowego elementu zapisywane są poszczególne znaki wchodzące w skład tekstu - na końcu znajduje się znak NULL - '\0' określający koniec tekstu.

Rzeczywisty rozmiar tablicy jest więc zawsze o jeden większy niż tekst w niej przechowywany. W języku C istnieje również możliwość definiowania stałych stanowiących łańcuchy znakowe.

Przykładowe deklaracje stałych stanowiących łańcuchy zapisane w pamięci operacyjnej komputera:

```
char tekst[]="ala ma kota";
```

lub równoznaczny zapis:

```
char *tekst="ala ma kota";
```

Zapis liczb w C

liczby całkowite: 1, 12, 0x045c. W większości przypadków w programach używa się zapisu dziesiętnego liczb, istnieje jednak również możliwość zapisu liczb całkowitych w formacie szesnastkowym - z dodatkowym prefiksem "0x..."

liczby zmiennoprzecinkowe: 1.153, 1.5e-3, 1.5e5. Liczby zmiennoprzecinkowe zapisuje się z przecinkiem dziesiętnym, bądź w postaci: "WARTOŚĆ_LICZBY * $10^{POTĘGA}$ ", co w języku C trzeba napisać jako: "WARTOŚĆ_LICZBY$e^{POTĘGA}$".

II-3. Operatory

Operatory definiują określone operacje których można dokonać na zmiennych w języku C.
Mogą nimi być operacje arytmetyczne bądź logiczne.
Poniżej przedstawiono podstawowe operacje w języku C.

Operacje arytmetyczne

Dodawanie: c = a **+** b //do zmiennej 'a' dodajemy 'b', wynik zapisany zostanie w 'c'

Odejmowanie: c = a **-** b

Mnożenie: c = a ***** b

Dzielenie: c = a **/** b

Reszta z dzielenia całkowitego (modulo):
c = a**%**b // np. 22% 5 = 2

Przykład na stosowanie operatorów arytmetycznych - obliczenie pola powierzchni trójkąta:

```
float pole_pow;
float bok_A,wysokosc;
```
deklaracje zmiennych

```
bok_A=22.5;  // dlugosc podstawy trojkata
wysokosc=15;    //wysokosc trojkata
pole_pow=(float)(0.5*bok_A*wysokosc); //wynik - obliczone pole powierzchni = 168.75
```
dane do obliczeń

wykonane obliczenia-operacje arytmetyczne

Operatory jednoargumentowe

! zmienna - negacja zmiennej

~ zmienna - uzupełnienie jedynkowe (uzupełnienie jedynkowe realizuje negację wszystkich bitów zmiennej - każdy bit jest negowany)

++ zmienna - inkrementacja

-- zmienna - dekrementacja

Operatory jednoargumentowe działają tylko na jeden parametr. Oto przykład negacji zmiennej:

```
BOOL zmienna;
```
deklaracja zmiennej

```
zmienna=1;              //przypisanie wartosci 1 czyli tzw. TRUE
zmienna = !zmienna;     //negacja zmiennej
//wartosc zmiennej w tym miejscu programu juz jest rowna 0
```
operator jednoargumentowy negacji

Operatory przesunięcia

zmienna **>>** n - przesunięcie w prawo zmiennej o n bitów

zmienna **<<** n - przesunięcie w lewo

Operator przypisania

x**=**y - do zmiennej x zostaje przypisana wartość zmiennej y

Operacje na bitach

x **&** y - Iloczyn logiczny bitów zmiennych x i y

x **|** y - suma logiczna bitów zmiennych x i y

Operatory logiczne

x**&&**y - jeśli spełniony warunek x i warunek y - iloczyn logiczny (Uwaga! - nie mylić z iloczynem bitowym)

x**||**y - suma logiczna – jeśli spełniony warunek x lub warunek y

Operatory relacji

x **>** y - zmienna x większa od zmiennej y

x**<** y - zmienna x mniejsza od zmiennej y

x**>=**y - zmienna x większa lub równa zmiennej y

x**<=**y - zmienna x mniejsza lub równa zmiennej y

x**==**y - zmienna x równa zmiennej y

x**!=**y - zmienna x różna od zmiennej y

II-4. Instrukcje

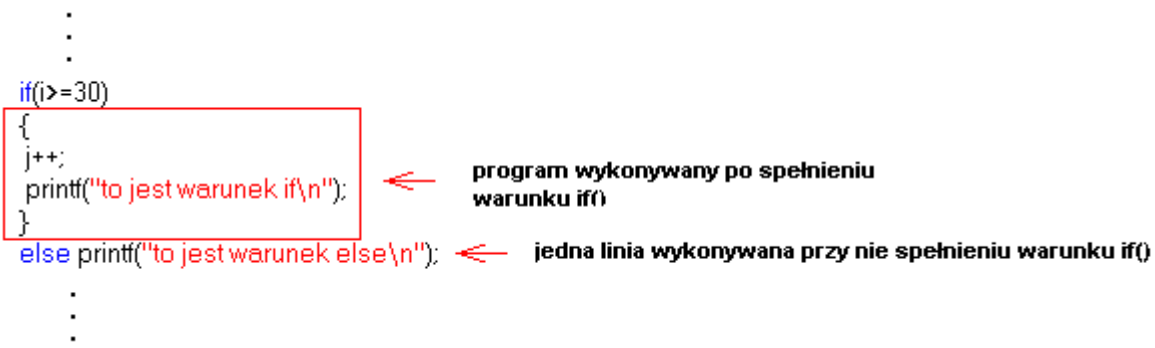
Instrukcja warunkowa ***if***

```
if(wyrażenie)
{
…….. ;
…….. ;
}
else
{
……;
……;
}
```

Instrukcja *if* stanowi często używaną instrukcję warunkową. Jeśli spełniony jest warunek funkcji, to wykonany zostanie fragment programu w nawiasie - po instrukcji *if*.

W przeciwnym wypadku realizowany jest program w nawiasie po instrukcji else. Instrukcja *else* jest opcjonalna.

Przykład instrukcji if():



Instrukcja wyboru ***switch***

```
switch(zmienna)
{
case wartość1 :    ……;
                   …….. ;
                   break;
case wartość2 :    ……;
                   …….. ;
                   break;
case wartość3 :    ……;
                   …….. ;
break;
}
```

Instrukcja *switch* stanowi funkcję wyboru. Wykonany zostanie jedynie fragment programu przyporządkowany do konkretnej wartości zmiennej wejściowej funkcji.

Przykład instrukcji *switch()*:

```
switch (i)
{
 case 0: printf("i jest równe 0 \n");
          break;

 case 1: printf("i jest równe 1 \n");
          break;

 case 2: printf("i jest równe 2 \n");
          break;

 default: printf("i jest domyslne - zadne z powyzszych \n");
          break;
}
```

dla kolejnych wartości zmiennej i wykonywane są kolejne fragmenty programu umieszczone po instrukcjach case

Pętla programowa ***for***

```
for ( warunek_start ; warunek_stop ; wyrażenie)
{
……..;
……..;
……..;
}
```

Instrukcja *for* stanowi pętlę programową, w której program wykonywany jest od spełnienia warunku startu, do momentu spełnienia warunku stop. Warunek ten stanowi najczęściej licznik inkrementowany żądaną ilość razy – tyle razy wykonany zostanie program w pętli *for*.

Przykład instrukcji *for()*:

```
for(i=0;i<=100;i++)
{
 printf("oto kolejne wywołania instrukcji if() \n");
 printf("numer wywolania=%d \n",i);
}
```

pętla będzie wywoływana wielokrotnie - zależnie od waruków Start, Stop

Pętla warunkowa ***while***

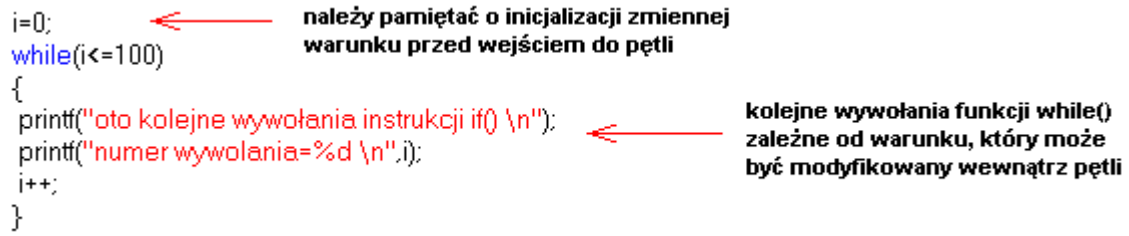
```
while(wyrażenie)
{
……..;
……..;
……..;
}
```

Pętla warunkowa *while* wykonuje powtórzenia programu znajdującego się między nawiasami do momentu spełnienia określonego wyrażenia (dopóki logiczna wartość wyrażenia jest różna od 0).

Wyrażenie to jest najczęściej modyfikowane wewnątrz pętli i jeśli jego wartość, po pewnej ilości obiegów, stanowi już wartość oczekiwaną jako zakończenie wykonywania programu, to następuje koniec wykonywania pętli programowej.

W pętli *while* warunek sprawdzany jest na początku pętli. Możliwa jest więc sytuacja, w której program wewnątrz pętli nie będzie wykonany ani razu – jeśli warunek jest spełniony już w momencie rozpoczęcia wykonywania pętli *while*.

Przykład wykorzystania pętli *while()*:

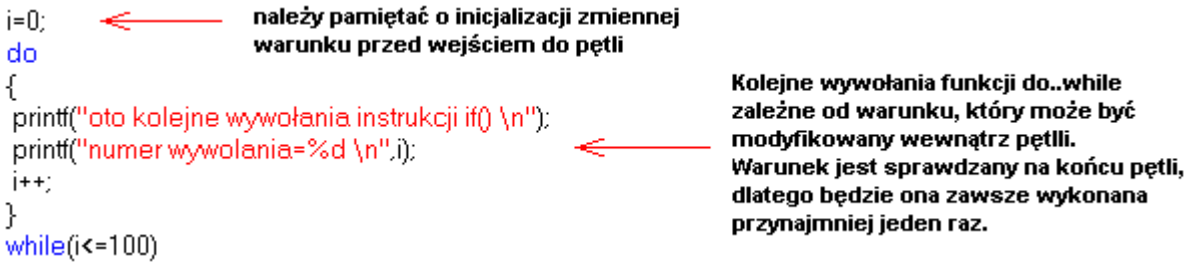


Pętla warunkowa ***do***

```
do
{
……..;
……..;
……..;
}
while(wyrażene)
```

Pętla *do .. while* jest w swoim działaniu podobna do pętli *while*. Różnicą jest to, że warunek zakończenia działania pętli jest sprawdzany na końcu pętli, czyli pętla ta będzie zawsze wykonana przynajmniej jeden raz.

Przykład wykorzystania pętli *do .. while()*:

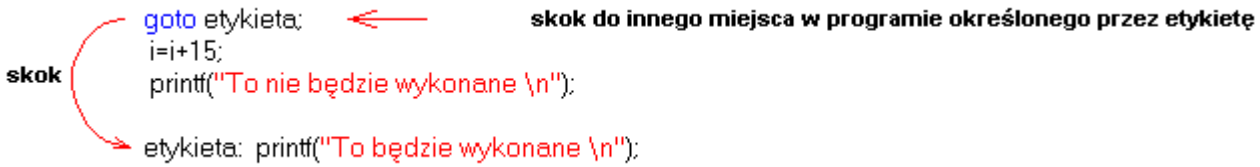


Instrukcja skoku do etykiety ***goto***

```
goto label1;
……..;
……..;
……..;
label1: ……;
```

Instrukcja *goto* stanowi instrukcję skoku bezwarunkowego do miejsca w programie określonego poprzez etykietę (nazwę) podawaną jako argument funkcji.

Przykład wykorzystania instrukcji *goto*:



II-5. Funkcje

Funkcje stanowią wydzielony fragment programu, który realizuje określone zadanie. Funkcje mogą być wywoływane wielokrotnie w programie poprawiając czytelność programu oraz redukując jego rozmiar.

Nie trzeba się już wtedy zagłębiać w sposób rozwiązania danego problemu, po prostu wywołuje się daną funkcję, w miejscach, w których istnieje potrzeba realizacji zadania wykonywanego przez tę funkcję.

Funkcje w języku C są definiowane w następujący sposób:

```
TypWartosciZwracanej NazwaFunkcji (TypParametru1 Par1, TypParametru2 Par2,….)
{
……;
……;
……;
……;
……;
return ZwracanaWartość;
}
```

Każda funkcja posiada swoją unikalną nazwę, określony zestaw parametrów oraz może zwracać wartość w wyniku swojego działania.

Oto przykład funkcji realizującej odejmowanie dwóch liczb:

```
float FunkcjaOdejmowania(float InA, float InB )
{
return (InA-InB);
}
```

Funkcja posiada swoją nazwę - "*FunkcjaOdejmowania",* dwa argumenty wejściowe - *"InA"* oraz *"InB"*, zwraca również wartość będącą w tym przypadku wynikiem odejmowania liczb stanowiących jej parametry wejściowe.

Tak zdefiniowaną funkcję można już wykorzystać we własnym programie - do realizacji odejmowania:

```
float wynik;
wynik=FunkcjaOdejmowania(5,2);        //wynik działania funkcji będzie równy 3
```

Argumenty funkcji przekazywane są do niej przez „*wartość”*, czyli do obliczeń wewnątrz funkcji jest przekazywana wartość zmiennej a nie jej adres.

Po wykonaniu obliczeń i opuszczeniu funkcji wartości zmiennych „*sprzed”* jej wywołania nie ulegają zmianie - chodzi o zmienne użyte jako argumenty funkcji.

Funkcja może również nie zwracać żadnej wartości w wyniku swojego działania. Typ zmiennej zwracanej jest wówczas zadeklarowany jako pusty - *void*. Deklaracja takiej funkcji będzie wtedy następująca:

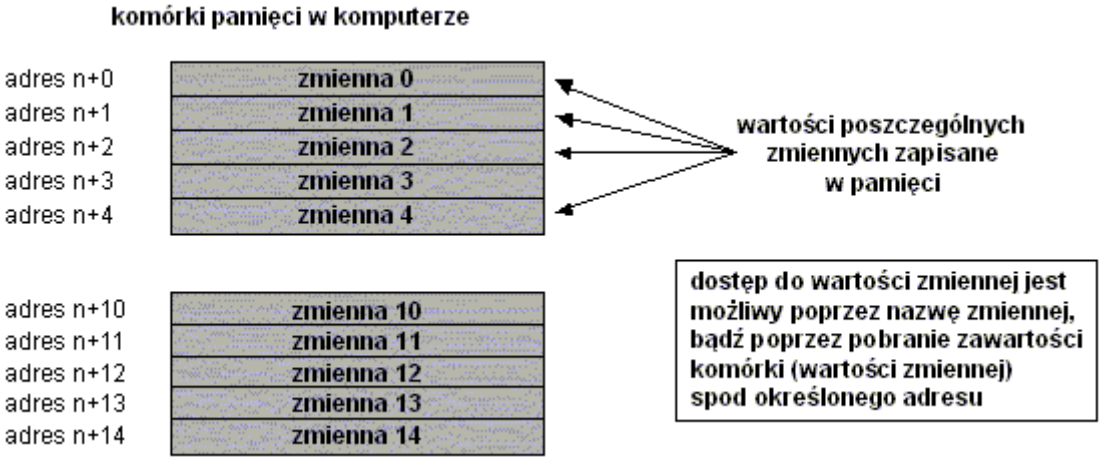
```
void NazwaFunkcji (TypParametru1 Par1, TypParametru2 Par2,….)
{
……;
……;
……;
……;
……;
}
```

II-6. Wskaźniki do zmiennych

W języku C istnieje możliwość pracy ze wskaźnikami, czyli można operować na adresach zmiennych. Zmienne stanowią pewne obszary pamięci zarezerwowane dla danych potrzebnych do realizacji obliczeń, bądź innych zadań programistycznych.

Ponieważ każda zmienna posiada określony rozmiar, który zajmuje ona w pamięci komputera (zależny od typu zmiennej), to pamięć komputera można sobie wyobrazić jako tablicę, w której w określonych miejscach znajdują się dane powiązane z określoną zmienną.

Położenie w pamięci - *"tablicy"* jest określone przez adres zmiennej. Do zawartości tej pamięci można się odwołać (pobrać wartość, bądź zapisać wartość do zmiennej) poprzez nazwę zmiennej, bądź poprzez jej położenie w pamięci - czyli adres.



W większości przypadków programista odwołuje się do wartości zmiennych poprzez ich nazwę. Nie korzysta wtedy z adresów, czyli nie używa operacji na adresach (wskaźnikach). Zdarzają się jednak sytuacje w których istnieje konieczność korzystania ze wskaźników, bądź korzystniejsze staje się użycie wskaźników.

Przykładem konieczności wykorzystania wskaźników jest omówiona dalej metoda zwrotu wartości poprzez argumenty wywołania funkcji. Funkcje w języku C mogą bezpośrednio zwrócić wyłącznie jedną wartość. Jeżeli istnieje konieczność przekazania więcej niż jednej wartości do programu wywołującego daną funkcję, to funkcja ta zwraca wyliczone wewnątrz wartości pod konkretne adresy zmiennych - podawane w wywołaniu tej funkcji jako argumenty.

Często też operacje na tablicach stają się znacznie łatwiejsze w implementacji programowej w przypadku korzystania ze wskaźników, niż operacjach indeksowych, ale tutaj już pozostaje dowolność realizacji określonych zadań zależna od wygody programisty.

Definicja wskaźnika do zmiennej określonego typu jest realizowana przy pomocy zapisu:

```
typ_zmiennej_wskazywanej *zmienna_wskaznik;
//zmienna_wskaźnik jest tutaj nazwą zmiennej, która stanowi wskaźnik na zmienną
//o typie typ_zmiennej_wskazywanej
```

Zmienna *zmienna_wskaznik* może przechowywać adres dowolnej innej zmiennej o typie *typ_zmiennej_wskazywanej*.

Znak * w definicji zmiennej określa w języku C, że zmienna z którą jest on skojarzony będzie wskaźnikiem - będzie mogła przechowywać adresy.

Przykład wykorzystania wskaźników:

```
int x,y;          //definicja dwóch zmiennych x,y typu integer
int *pointer;     //definicja wskaźnika na zmienną typu integer – czyli zmienna
                  //pointer będzie zawierać adres dowolnej zmiennej typu integer
pointer = &x;     //pobrany zostaje adres zmiennej x i zapamiętany w zmiennej pointer
y=*pointer;       //pobrana zostaje zawartość zmiennej, której adres znajduje się
                  //we wskaźniku pointer
```

W powyższym zadaniu dokonano kopiowania stanu zmiennej x do y na podstawie adresów zmiennych.

Znak **&** w operacji przypisania (zob. powyżej - *pointer = &x*) określa w języku C, że pobrany będzie adres zmiennej określonej za znakiem &.

Do wskaźnika nie będzie więc pobrana aktualna wartość zmiennej lecz jej adres.
Znak * w operacji przypisania (zob. powyżej - *y=*pointer*) określa w języku C, że do zmiennej y przypisana zostanie wartość zmiennej o adresie przechowywanym w zmiennej określonej za znakiem *.

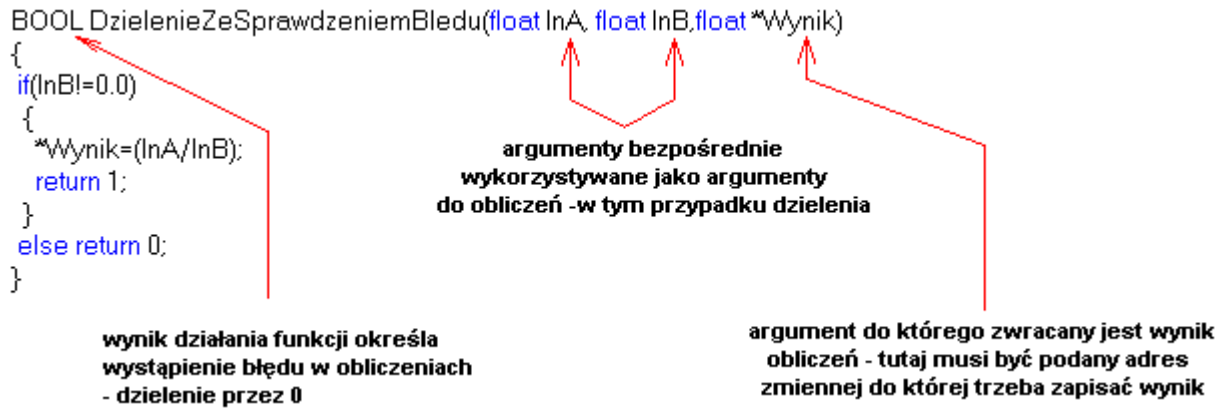
Powyższy przykład odpowiada prostemu przypisaniu zrealizowanemu przy pomocy operacji na wskaźnikach:

```
y=x;
```

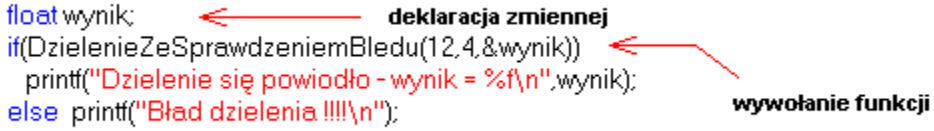
W rozdziale dotyczącym funkcji zostało powiedziane, że **funkcje** w języku C pobierają do obliczeń wewnątrz jedynie wartości argumentów a nie ich adresy. Bezpośrednio nie ma więc możliwości przekazania wyników obliczeń wewnątrz funkcji do zmiennych na zewnątrz.
W sposób natychmiastowy można z funkcji otrzymać tylko jeden rezultat stanowiący wynik działania funkcji.

Jeśli istnieje konieczność zwrócenia więcej niż jednej wartości jako rezultatu działania funkcji to, jako argument należy przekazać adres (wskaźnik) zmiennej zewnętrznej, której wartość trzeba zwrócić. Wartość wyliczona wewnątrz funkcji jest wtedy zwracana pod konkretny adres - zmiennej zewnętrznej.

Poniżej przedstawiono przykład definicji funkcji - dzielenie ze sprawdzeniem błędu dzielenia przez zero:



Wywołanie takiej funkcji we własnym programie powinno wyglądać następująco:



Częstym zastosowaniem wskaźników są operacje na **tablicach**. Do elementów tablicy można się odwoływać zarówno przy pomocy indeksu (numeru elementu w tablicy), jak i przy pomocy wskaźnika na dany element tablicy. Odwołanie do nazwy tablicy w języku C określa odwołanie do pierwszego elementu tablicy. W języku C numery elementów tablicy zaczynają się od 0.

Przykładowe operacje na tablicach:

```
int tablica[10];        //definicja tablicy ze zmiennymi typu integer o rozmiarze 10
int *pointer;           //definicja wskaźnika do zmiennej typu integer

int zmienna_int;        //definicja zmiennej typu integer
```

Jeśli istnieje konieczność pobrania konkretnego elementu tablicy (np. 5 elementu), to można tego dokonać na dwa sposoby:

```
zmienna_int = tablica[4];   //zapis indeksowy - należy pamiętać,
                            //że elementy tablicy są liczone od 0

zmienna_int = *(tablica+4);         //zapis wskaźnikowy;
                                    //* oznacza pobranie zawartości spod danego adresu
                                    //(odwołanie do nazwy tablicy określa adres
                                    //elementu o indeksie 0)
                                    //do tego elementu dodawane jest jeszcze przesunięcie
```

Jeśli istnieje konieczność pobrania adresu danego elementu tablicy (pobranie wskaźnika na dany element tablicy), np. pierwszego elementu to można to wykonać znów na dwa sposoby:

```
pointer = &tablica[0];          //zapis indeksowy - należy pamiętać, że elementy
                                //tablicy są liczone od 0

pointer = tablica;              //zapis wskaźnikowy
```

Należy tutaj zwrócić uwagę na to, że choć odwołanie do nazwy tablicy określa odwołanie do elementu spod adresu 0 w tablicy (ma więc znaczenie wskaźnika do zmiennej), to tablica nie stanowi zmiennej wskaźnikowej. Nie będą więc tutaj dozwolone operacje typowe dla wskaźników - inkrementacja, dekrementacja czy przypisanie nowej wartości do wskaźnika.

Prawidłowym zapisem będzie:

```
pointer = tablica;      //pobranie adresu elementu o indeksie 0 tablicy do wskaźnika
```

Nieprawidłowym zapisem jednak będzie zapis odwrotny:

```
tablica = pointer;      //zmienna tablica nie jest wskaźnikiem - zapis jest nieprawidłowy
```

II-7. Wybrane funkcje biblioteczne

Funkcje ***printf() scanf()***

W języku C istnieje możliwość pobrania danych ze standardowego wejścia oraz wysłania danych do standardowego wyjścia (w najprostszym przypadku może to być klawiatura i ekran).

Do wyświetlania zmiennych służy funkcja *printf()*.

Przykładowo aby wyświetlić napis „*ala ma kota”* należy zapisać operację:

```
printf(”ala ma kota”);
```

Funkcja printf() może powodować wyświetlanie zmiennych różnego typu, wówczas trzeba przekazać do niej informacje na temat zmiennej oraz typu tej zmiennej:

```
float zm1;
zm1=2.376;
printf(”Zmienna 1: %06.2f”,zm1);
```

Spowoduje to wyświetlenie napisu:

Zmienna 1: 002.38

%06.2f - jest tutaj identyfikatorem formatu zmiennej

06	cała liczba zajmuje 6 miejsc, w razie konieczności zostaje ona uzupełniona zerami z przodu (jeśli zajmuje mniej niż 6 pozycji)
2	dwa miejsca po przecinku
f	typ zmiennej (float).

Inne identyfikatory typu:

%d	argument przekształcony zostaje do postaci dziesiętnej
%x	postać szesnastkowa
%u	postać dziesiętna bez znaku
%s	tekst
%c	pojedynczy znak tekstowy.

W funkcjach *printf() scanf()* można również używać specjalnych znaków sterujących:

\n	nowy wiersz
\t	tabulacja
\b	cofanie
\r	powrót kursora na początek linii
\f	nowa strona
****	„backslash”
\’	apostrof

Funkcje ***sprintf() sscanf()***

Jeśli za wejście, bądź wyjście przyjmie się zmienną typu znakowego, to do realizacji powyższych zadań służą analogiczne funkcje: *sprintf()* oraz *sscanf()*.

Funkcje te zamiast na ekran będą przekazywać dane do łańcuchów tekstowych.

Przykład:

```
char chr[250];
float zm; zm=2.984;
sprintf(chr,"%06.2f",zm);
```

Powyższe instrukcje spowodują, że do zmiennej tekstowej chr zostaje zapisana wartość zmiennej zm typu float.

Podobna realizacja w przypadku funkcji sscanf działa w odwrotną stronę:

```
float val;
char *chr="123";
sscanf(chr,"%f",&val);
```

Jeśli w tekście chr znajduje się poprawna wartość dziesiętna w formacie zmiennoprzecinkowym zapisana w postaci tekstu, to zostaje ona przekształcona do typu float i przepisana do zmiennej val.
Należy zwrócić uwagę na argument funkcji *sscanf* - val, ponieważ funkcja do tego argumentu zwraca wartość, to w wywołaniu musi być podany adres zmiennej typu float, do której ma być wyprowadzony rezultat (znak & przed nazwą zmiennej).
Przy wywołaniu funkcji *sprintf* zaistniała podobna sytuacja, ale tam argumentem do którego zwracano rezultat działania funkcji była tablica znaków. Nazwa tablicy jest zaś jednocześnie wskaźnikiem do pierwszego elementu tablicy - czyli tutaj również była dokonywana operacja na adresach.
Operacje na wskaźnikach dokładniej wyjaśnione zostały w rozdziale *"Wskaźniki do zmiennych"*.

Funkcje ***strcpy strlen***

Funkcje te należą do rodziny funkcji realizujących operacje na łańcuchach tekstowych.

*strcpy (char *ŁancuchDocelowy,*ŁańcuchŹródłowy)*
realizuje kopiowanie zawartości jednego z łańcuchów tekstowych do drugiego łańcucha

*int strlen(char *ŁańcuchTekstowy)* zwraca długość tekstu w łańcuchu tekstowym.

Funkcje ***fopen fprintf fscanf fclose***

Funkcje te służą do obsługi plików.

*FILE *fopen(char *NazwaPliku, char *Atrybuty)* - otwarcie pliku z odpowiednimi atrybutami np.:

- **r** tylko do odczytu
- **w** tylko do zapisu
- **a** do zapisu - tworzenie nowego pliku tylko gdy taki plik jeszcze nie istnieje
- **t** tryb tekstowy pliku.

*fprintf(FILE *WskaźnikDoPliku, char *DefinicjaFormatu, argumenty)* - analogiczna funkcja do *printf*, ale służąca do zapisu danych do pliku

*fscanf(FILE *WskaźnikDoPliku, char *DefinicjaFormatu, argumenty)* - funkcja analogiczna do *scanf* ale służąca do odczytu informacji z piku

*fclose(FILE *WskaźnikDoPliku)* - zamknięcie pliku.

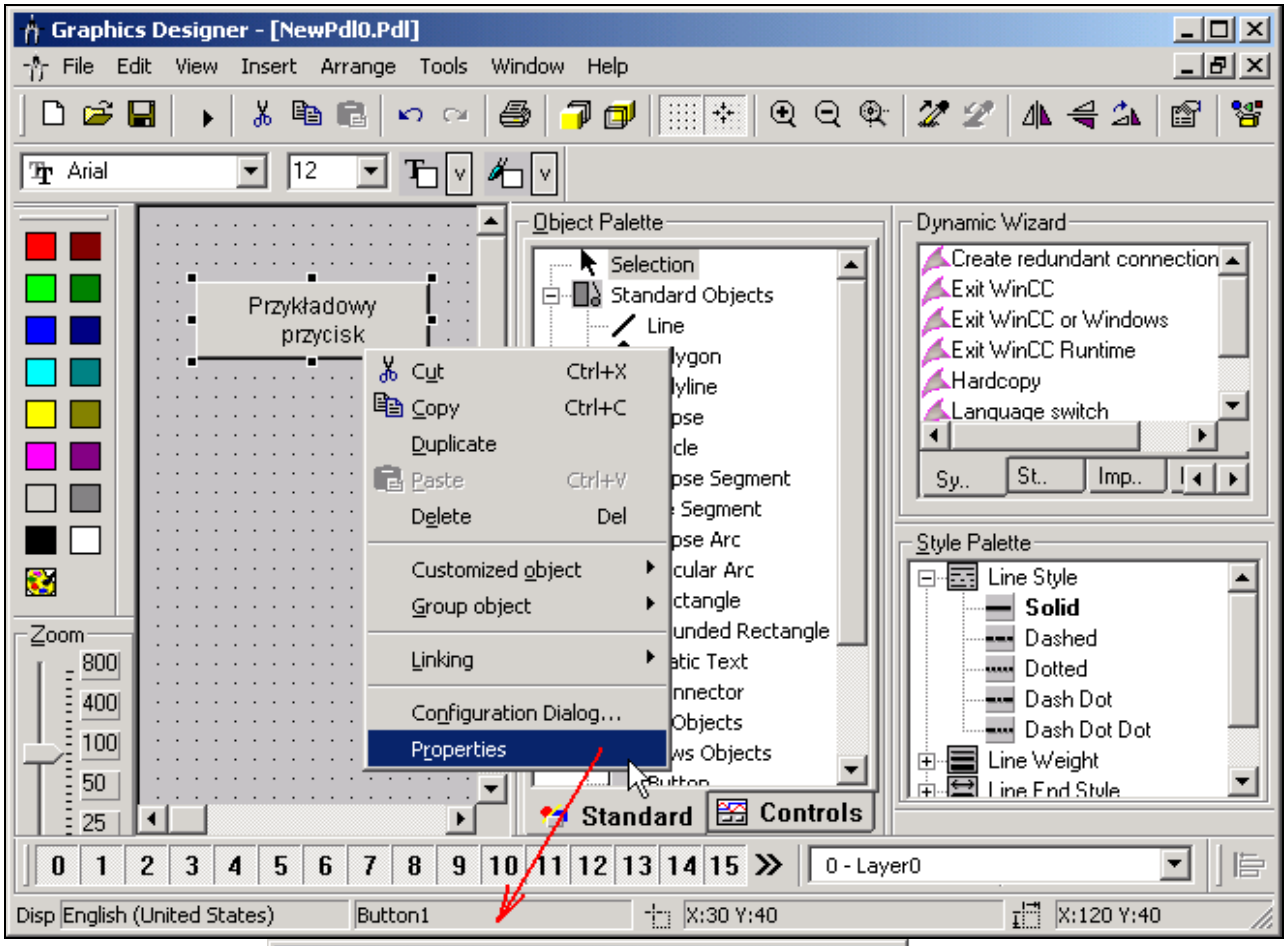
Przykładowy zapis do pliku:

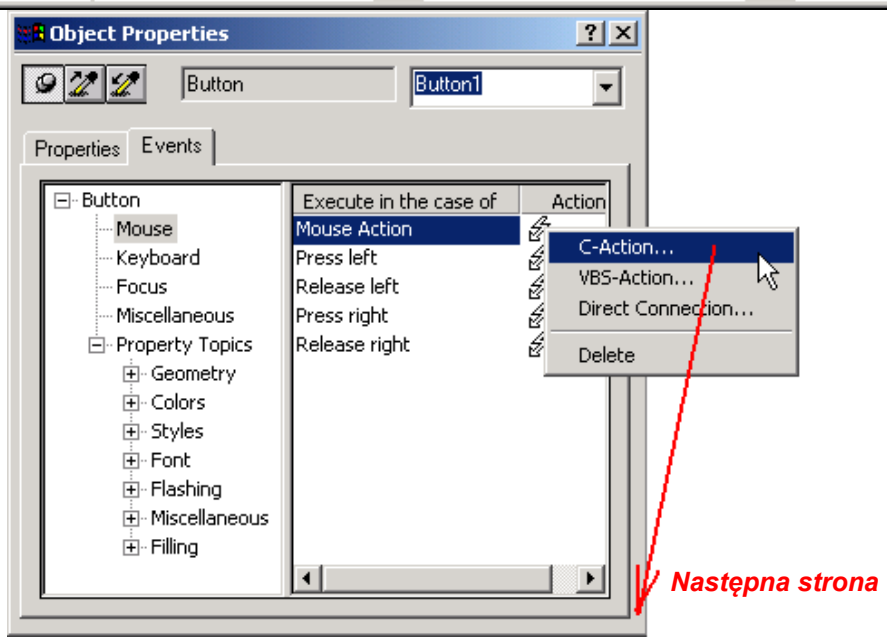
```
FILE *pFile;  //deklaracja zmiennej - uchwytu pliku


pFile=fopen("C:\\test1.txt","at"); //otwarcie pliku : a - jeśli plik istnieje, to jego zawartość
                                   //nie ulegnie zniszczeniu będzie on jedynie "otwarty do
                                   //edycji", t - plik będzie plikiem tekstowym


fprintf(pFile,"Ala ma kota"); //wysłanie przy pomocy funkcji analogicznej do printf
                              //łańcucha tekstowego "Ala ma kota" do pliku

fclose(pFile); //zamknięcie pliku
```

III. Ćwiczenia praktyczne

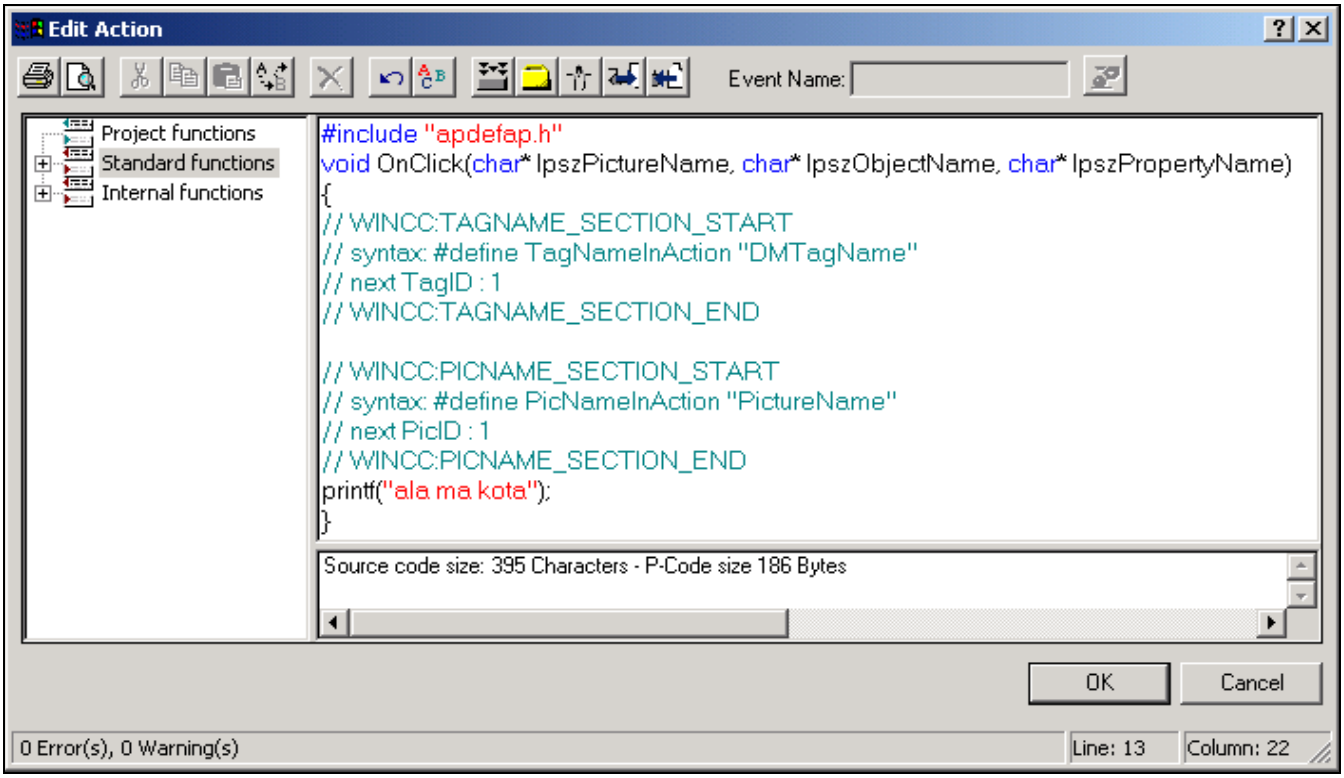
III-1. Mój pierwszy program

Skrypty w *WinCC* w większości przypadków działają w sposób zdarzeniowy - określone zdarzenie, (np. *kliknięcie* w obszar obiektu graficznego wizualizacji) powoduje wygenerowanie skryptu skojarzonego ze zdarzeniem myszy.
W *WinCC* skrypty takie nazywają się *"akcjami"* i stanowią określoną funkcję zapisaną przy użyciu edytora skryptów. Poniżej pokazano sposób edycji przykładowej funkcji powiązanej z przyciskiem w edytorze *WinCC*:





Następna strona



Widać, że akcja jest funkcją napisaną w języku C z odpowiednimi parametrami wejściowymi.

W *WinCC* dostępne jest okienko, do którego można wysyłać teksty z poziomu tworzonego programu skryptowego.

Dla *WinCC* jest ono odpowiednikiem ekranu monitora w trybie tekstowym i służy tutaj głównie do celów diagnostycznych.

Można na nie wysłać np. wartości zmiennych wyliczanych wewnątrz funkcji - w ten sposób dowiadując się o ewentualnych błędach w pisanych programach.

Zmienna - obszar pamięci, w którym przechowywane są wartości danych potrzebnych do obliczeń. Do zmiennej można odwołać się poprzez jej nazwę *(zob. "Zmienne").*
Funkcja - fragment programu realizujący określone zadanie *(zob. "Funkcje").*

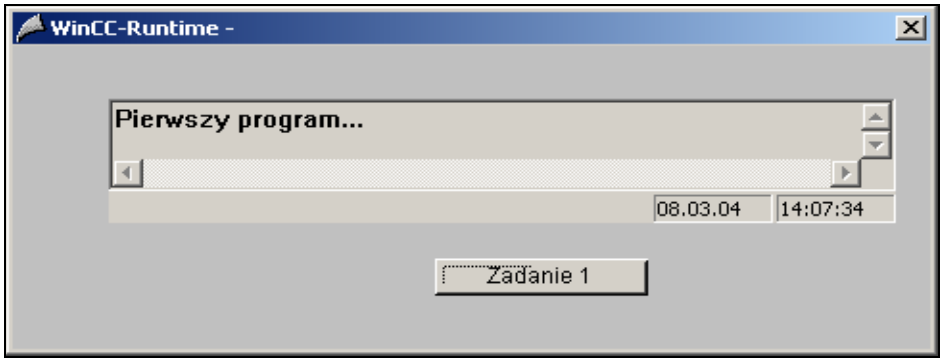
W pierwszym zadaniu trzeba będzie wysłać na ekran (w przypadku *WinCC* na obszar okna diagnostycznego) tekst *"Pierwszy program ...."*. Można tego dokonać przy wykorzystaniu funkcji *printf()* służącej do formatowania tekstów a następnie wyświetlania ich na ekranie komputera *(zob. opis funkcji printf w rozdziale "Funkcje biblioteczne")*.

Przykładowy program może być następujący:

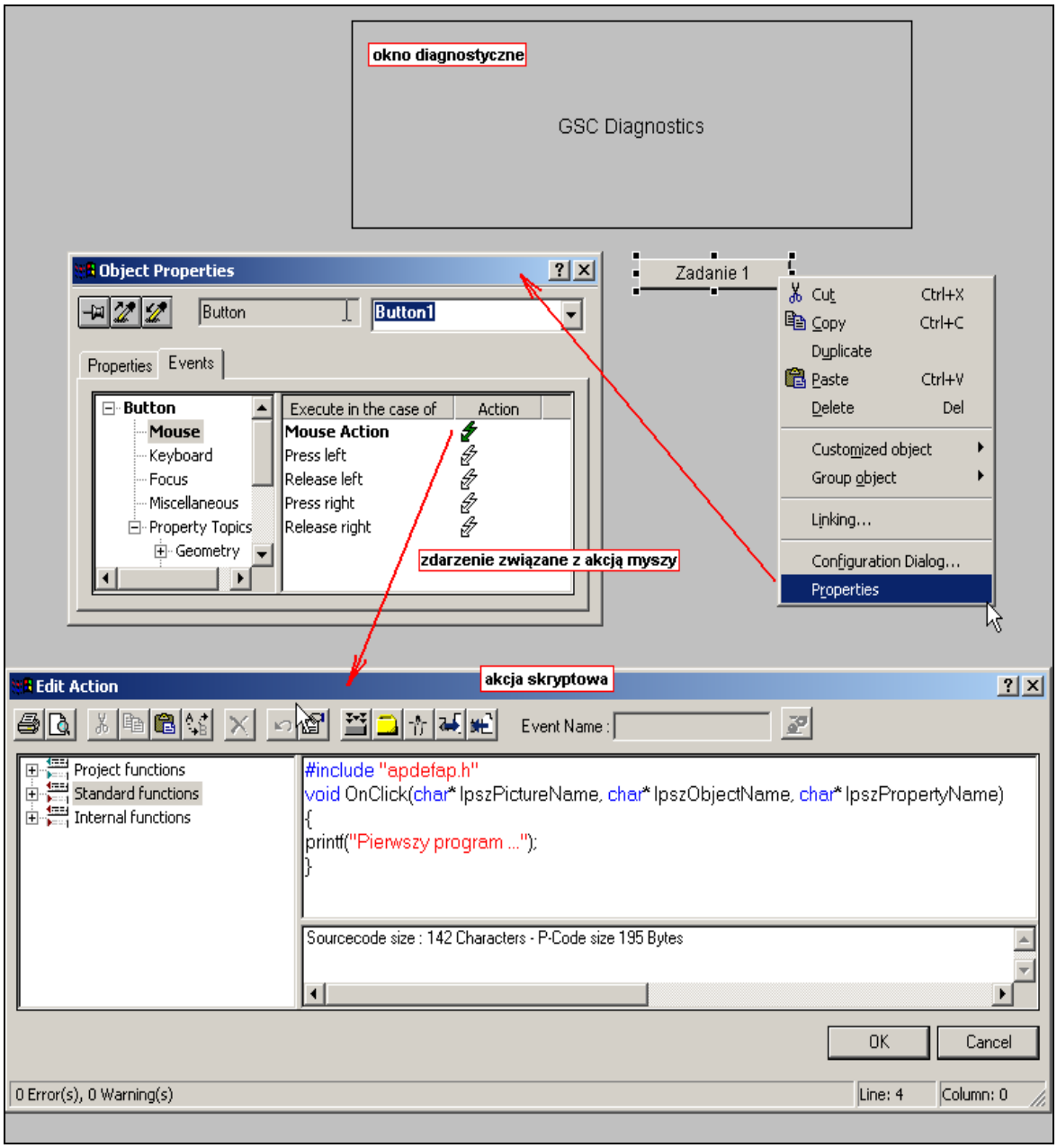
```
printf("Pierwszy program ...");
```

Nie należy zapominać tutaj o odpowiedniej składni języka C *(zob. "Struktura jezyka C").*

Spowoduje to wyświetlenie napisu w oknie diagnostycznym:

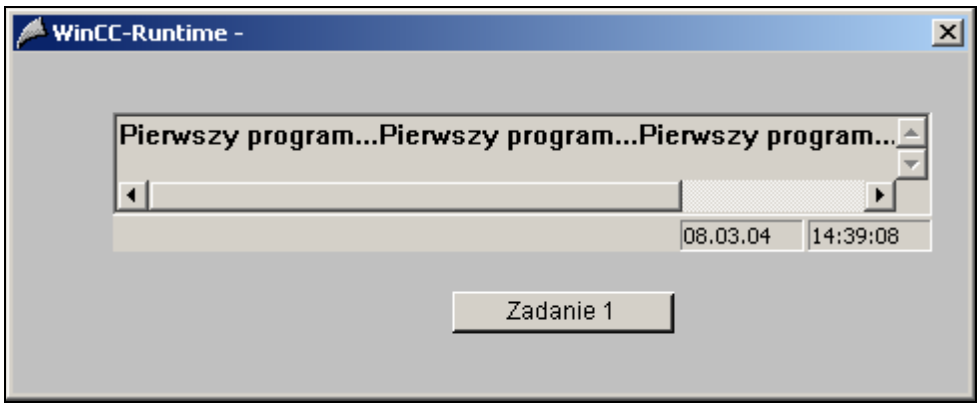


Program ten jest skojarzony z przyciskiem *"Zadanie 1"* i wywoływany po jego naciśnięciu. W *WinCC* istnieje możliwość skojarzenia programu z określonym zdarzeniem - tutaj z akcją myszy. W zdarzeniu tym określona jest funkcja, w której zapisano powyższy program.



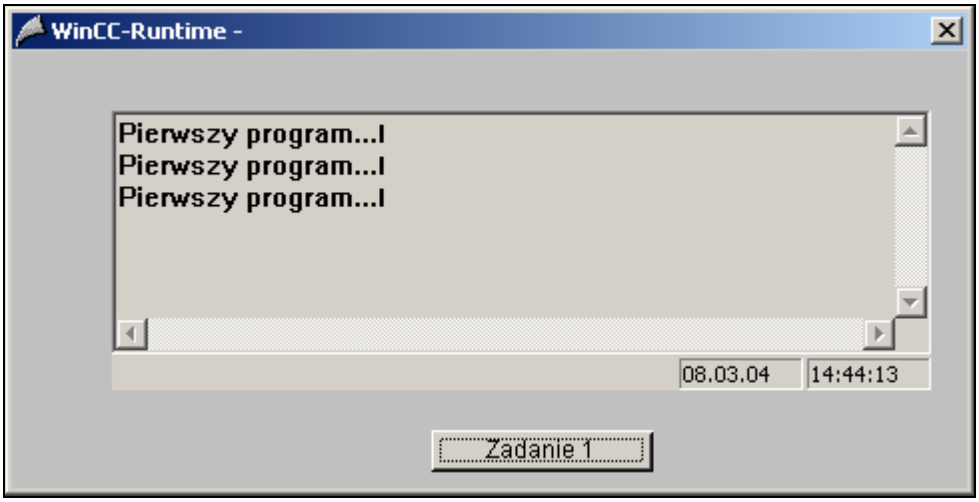
Zadania omawiane w niniejszym podręczniku zawsze będą pisane w postaci programów wywoływanych naciśnięciem przycisku.

Wyniki operacji będą wyświetlane w oknie diagnostycznym.
Kolejne naciśnięcia przycisku spowodują wyświetlenie kolejnych tekstów obok siebie:



Jeśli programista oczekuje wyświetlenia kolejnych informacji w kolejnych liniach na ekranie, to trzeba dodać do wywołania funkcji *printf()* odpowiedni znak formatujący określający koniec linii *(zob. opis funkcji printf w rozdziale "Funkcje biblioteczne")*.

Uzyska się wtedy czytelną informację - kolejne teksty będą wyświetlone w kolejnych liniach:
Program realizujący to zadanie: *printf("Pierwszy program ...\n");*



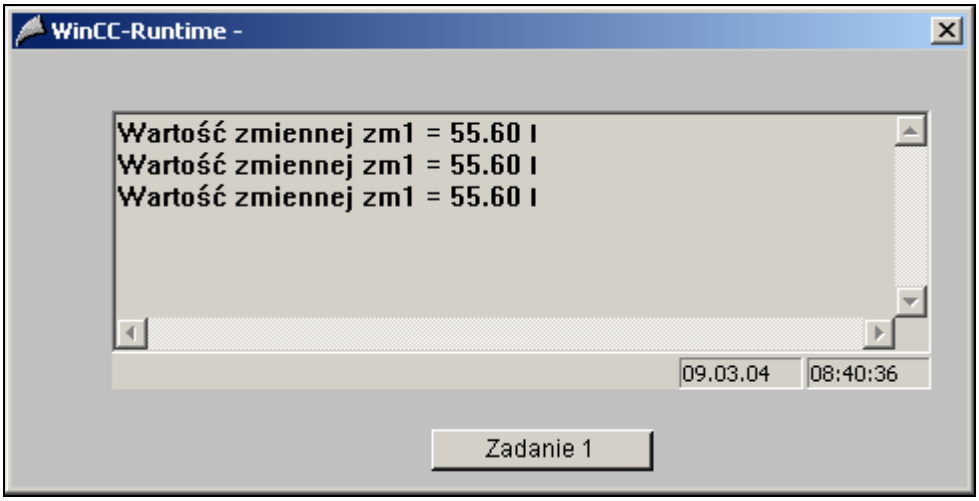
Kolejnym rozwinięciem wykorzystania funkcji *printf()* może być próba wyświetlenia wartości zmiennej. Do tego celu znowu trzeba użyć określonego znaku formatującego dotyczącego typu zmiennej *(zob. opis funkcji printf() w rozdziale "Funkcje biblioteczne")*.

Jeśli operacje będą wykonywane na zmiennych typu *float (zob. "Zmienne")*, czyli prezentujących typ zmiennoprzecinkowy, to znakiem formatującym będzie *"%f"*. W zadaniu powinna wystąpić zmienna o nazwie *zm1*, której przypisano określoną wartość - np. *55.6*.
Teraz należy wartość tej zmiennej wyświetlić na ekranie.

Zadanie to może realizować poniższy skrypt:

```
float zm1;      //deklaracja zmiennej typu float o nazwie zm1
                //(zob. deklaracje zmiennych w rozdz. "Zmienne")
zm1=55.6;       //przypisanie wartości 55.6 do zmiennej (zob. "Operatory w C")
printf("Wartość zmiennej zm1 = %5.2f\n",zm1);
                //wysłanie odpowiednio sformatowanego tekstu na ekran
                //tekst będzie zawierał również aktualną wartość zmiennej zm1 w formacie
                //float
                //cała szerokość pola liczby będzie wynosić 5 pozycji z czego 2 będą po
                //przecinku
                //(zob. opis funkcji printf w rozdziale "Funkcje biblioteczne")
```

Wynik działania skryptu:



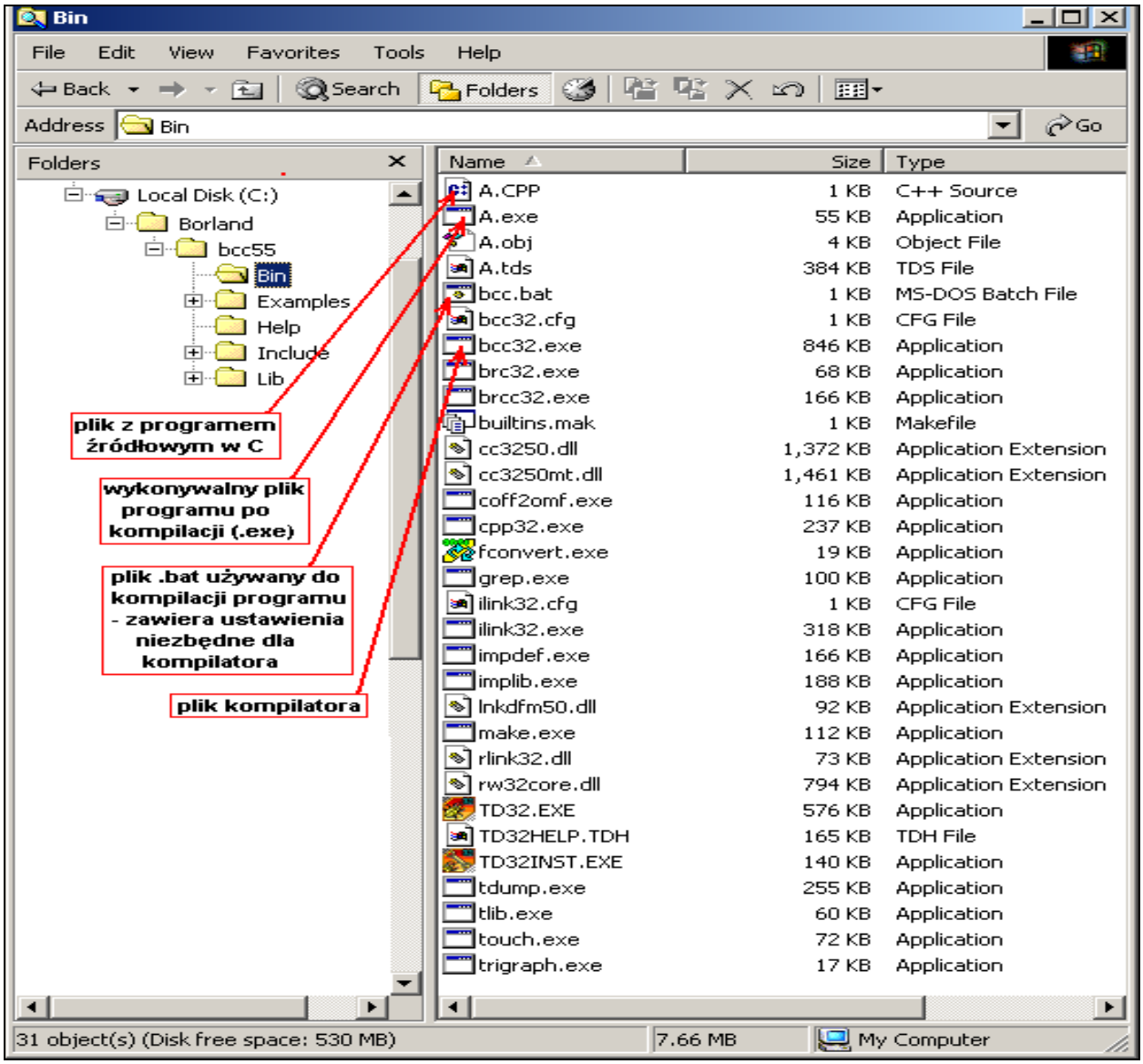
W ten sposób pokazano wykorzystanie funkcji *printf()* do formatowania tekstów wysyłanych na ekran.

Przykłady z podręcznika można próbować przećwiczyć samodzielnie wykorzystując jeden z darmowych kompilatorów pobranych z Internetu.

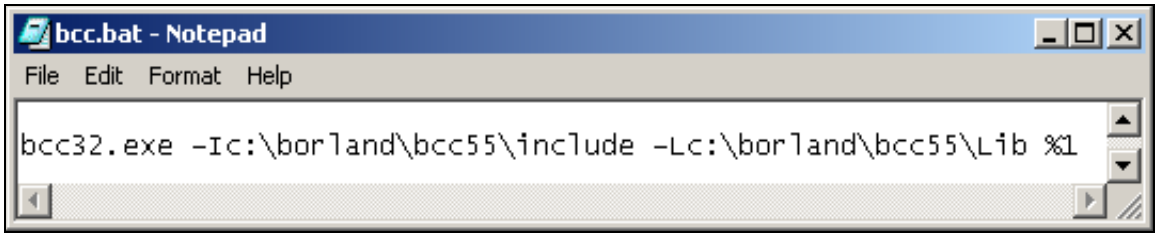
Jednym z najczęściej stosowanych kompilatorów jest *Borland C++ Compiler (np. wersja 5.5)*. Można go znaleźć w Internecie wpisując w wyszukiwarce internetowej hasło *"bcc32.exe"* - nazwa pliku wykonywalnego kompilatora.

Poniżej przedstawiono sposób kompilacji programu wynikowego przy pomocy tegoż właśnie kompilatora:

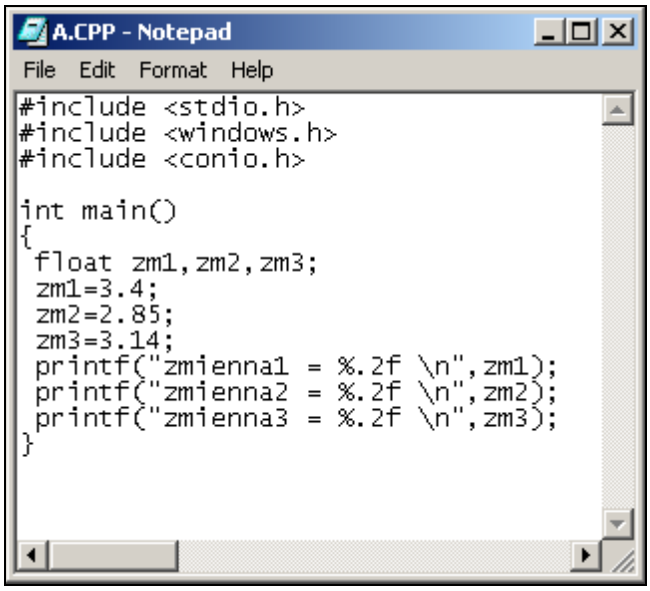
1. skopiowanie plików kompilatora do odpowiedniego katalogu (w omówieniu założono, że pliki będą się znajdowały w katalogu *"c:\borland\"* - ścieżka będzie istotna dla kompilacji programu



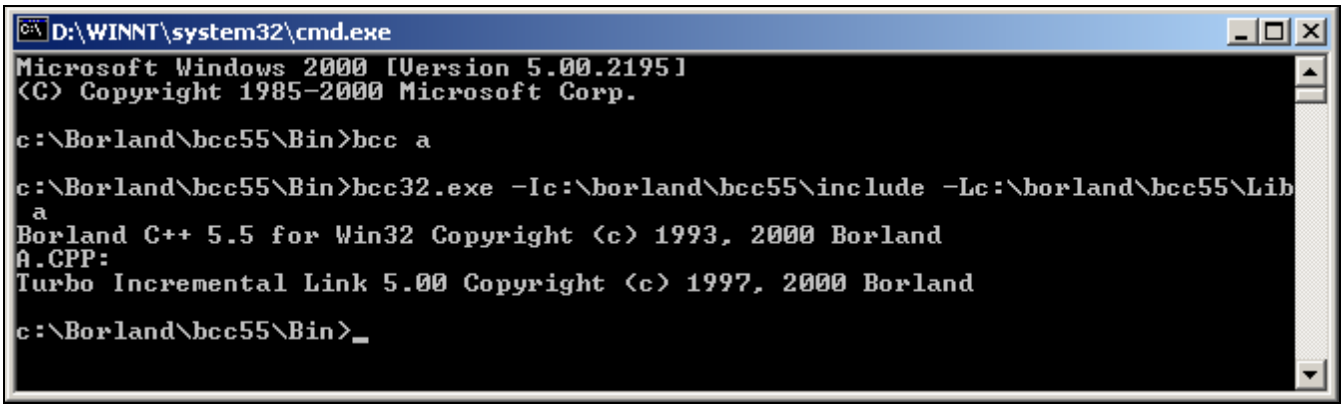
2. przygotowanie odpowiedniego pliku *".bat"* ułatwiającego kompilację programów - *"bcc.bat"*.



3. przygotowanie pliku z programem (plik musi posiadać rozszerzenie *".cpp"* np. *"a.cpp"* i musi znaleźć się w odpowiednim katalogu - w tym samym co plik kompilatora):



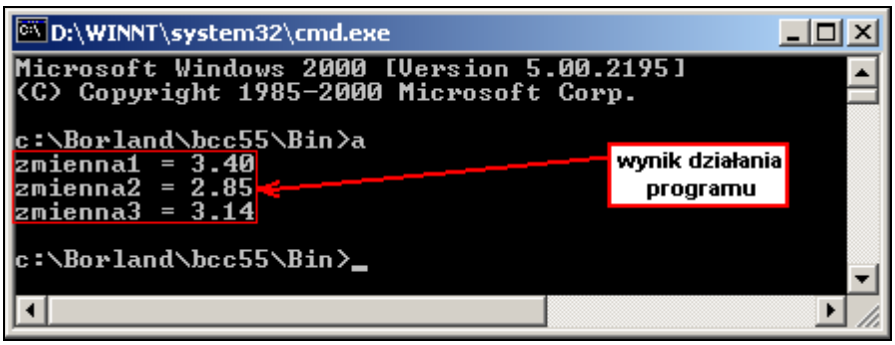
4. przejście do wiersza poleceń, w którym najpierw będzie dokonywana kompilacja programu, a następnie uruchamiany program wynikowy. Kompilacja programu z linii komend - *"bcc a",bcc* - plik *".bat"* z ustawieniami kompilatora - *"bcc.bat"; a* - plik z kodem źródłowym programu, który ma być skompilowany - *"a.cpp"*:



5. jeśli w wyniku kompilacji nie zostały zarejestrowane błędy, to automatycznie zostanie uruchomiony *linker* - program generujący na podstawie plików uzyskanych w trakcie kompilacji plik wynikowy typu *".exe"*, który stanowi już wykonywalny kod kompilowanego programu.

   Sprawdzenie działania programu polega na uruchomieniu pliku *".exe"* z linii komend. Kompilowane programy są generowane jako programy pracujące w trybie tekstowym - w konsoli *Windows,* ale w zupełności wystarczą do sprawdzenia działania przykładów realizowanych w niniejszym podręczniku.

6. uruchomienie programu *".exe"* z linii komend celem sprawdzenia jego działania:



W programach napisanych w kompilatorach języka C musi znaleźć się funkcja ***main()***, która stanowi główną funkcję programu. Właśnie ta funkcja jest wykonywana przez program napisany w języku C i przykładowe programy należy umieszczać wewnątrz funkcji *main()*.

Do poprawnego działania programów niezbędne będą również odpowiednie **pliki nagłówkowe** zawierające informacje o podstawowych typach danych i zmiennych wykorzystywanych w prezentowanych przykładach *(zob. "Struktura języka C")*:

```
#include <stdio.h>
#include <windows.h>
#include <conio.h>
int main()
{
....
....
}
```

Na zakończenie rozdziału krótki test sprawdzający.

Zadanie

Należy w kolejnych liniach wyświetlić na ekranie wartości zmiennych typu *float*: *zm1, zm2, zm3*. Zmienne mają być prezentowane na 2 miejscach po przecinku i opatrzone dodatkowym opisem. Które z prezentowanych rozwiązań jest prawidłowe ? Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
float zm1,zm2,zm3;
zm1=3.4;
zm2=2.85;
zm3=3.14;
printf("zmienna1 = %.2f \n",zm1);
printf("zmienna2 = %.2f \n",zm2);
printf("zmienna3 = %.2f \n",zm3);
```

Rozwiązanie 2

```
float zm1,zm2,zm3;
zm1=3.4;
zm2=2.85;
zm3=3.14;
printf("zmienna1 = %.2f \n");
printf("zmienna2 = %.2f \n");
printf("zmienna3 = %.2f \n");
```

Rozwiązanie 3

```
float zm1,zm2,zm3;
zm1=3.4;
zm2=2.85;
zm3=3.14;
printf("zmienna1 = %f \n",zm1);
printf("zmienna2 = %f \n",zm2);
printf("zmienna3 = %f \n",zm3);
```

III-2. Podstawowe operacje matematyczne

Język programowania umożliwia dokonywanie obliczeń matematycznych. Jest to jedno z najczęstszych zastosowań języka C w przypadku skryptów w *WinCC*. Na początek zostaną wykonane operacje typu: dodawanie i odejmowanie *(zob. "Operatory")*.
Proponowane rozwiązanie:

```
float zm1,zm2,wynik;        //definicja zmiennych wykorzystywanych w programie

zm1=2;                      //przypisanie wartości do zmiennych wykorzystywanych
                            //w trakcie obliczeń
zm2=3;
wynik=zm1+zm2;              //wykonanie określonego obliczenia
printf("wynik= %f",wynik);  //wyświetl wynik na ekranie
```

Widać, że w propozycji rozwiązania zadawano stałe wartości do obliczeń. Wartości zmiennych w sytuacjach praktycznych najczęściej stanowią jakieś dane, na których istnieje potrzeba wykonania określonych przeliczeń.

Jeszcze tylko próba sprawdzenia wiedzy na przykładzie krótkiego testu. Które z prezentowanych rozwiązań jest prawidłowe (odpowiedź znajduje się na końcu podręcznika)?

Zadanie

Napisać funkcję - *Funkcja1* realizującą dodawanie na dwóch argumentach typu *float* InA i InB *(zob."Funkcje")*. Funkcja powinna zwracać wynik działania. Które z prezentowanych rozwiązań jest prawidłowe ? Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
float InA,InB,Wynik;
InA=3.4;
InB=2.85;
Wynik=InA+InB;
```

Rozwiązanie 2

```
void Funkcja1(float InA, float InB)
{
return (InA+InB);
}
```

Rozwiązanie 3

```
float Funkcja1(float InA, float InB)
{
return (InA+InB);
}
```

III-3. Wykorzystanie instrukcji if()

Instrukcja warunkowa *if()* służy do uzależnienia wykonywania fragmentów programu od spełnienia określonego warunku *(zob. "Instrukcje w jezyku C")*.

Zadanie będzie polegało na wysłaniu określonego napisu na ekran.
Napis będzie odzwierciedlał stan zmiennej:

- Dla wartości zmiennej mniejszej lub równej 10 napis będzie brzmiał:
  "Za niska wartość zmiennej: xxx", gdzie *xxx* określa aktualną wartość zmiennej.

- Dla wartości od 11 do 89 napis będzie brzmiał:
  "Wartość zmiennej w dobrym przedziale: xxx".

- Dla wartości zmiennej równej i powyżej 90 napis będzie brzmiał:
  "Za wysoka wartość zmiennej: xxx".

Propozycja rozwiązania:

```
float zm1;      //deklaracja zmiennej zm1

zm1=5;          //zadanie wartości zm1 do celów symulacji

                //instrukcja if dla wartości zm1 mniejszych i równych 10
if(zm1<=10) printf("Za niska wartość zmiennej: %.0f",zm1);

                //instrukcja if dla wartości zm1 od 10 do 90
if(zm1>10 && zm1<90) printf("Wartość zmiennej w dobrym przedziale: %.0f",zm1);

                //instrukcja if dla wartości zm1 większych i równych 90
if(zm1>=90) printf("Za wysoka wartość zmiennej: %.0f", zm1);
```

Widać, że zależnie od aktualnej wartości zmiennej *zm1* będzie wyświetlany jeden z napisów - zależny od spełnienia warunku w określonej instrukcji *if()*.

Zadanie

Jaka będzie wartość zmiennej *zm3* po wykonaniu poniższego programu ?
Prawidłowa odpowiedź znajduje się na końcu podręcznika.

```
float zm1,zm2,zm3;
zm1=15;
zm2=10;
zm3=1;
zm3=(zm1+zm2)*zm2;
if(zm1>250) zm3=zm2+10;
else
{
zm3=zm3+5;
}
```

III-4. Uniwersalna funkcja realizująca podstawowe operacje matematyczne

Zadanie polega na wykonaniu funkcji, która będzie posiadała 3 argumenty wejściowe:
InA typu *float*, *InB* typu *float* oraz *Operacja* typu *char (zob. "Zmienne").*
Zmienne *InA*, *InB* będą stanowiły argumenty, na których będą wykonywane operacje arytmetyczne.

Trzeci parametr wejściowy będzie określał rodzaj operacji:

+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie

Wynik będzie zwracany jako rezultat działania funkcji. W przypadku błędu operacji (dzielenie przez zero) funkcja będzie zwracać wartość 0.

Zadanie

Które z poniższych rozwiązań stanowi poprawny program opisanej funkcji ?
Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
void DzialanieArytmetyczne(float InA, float InB, char Dzialanie)
{
if(Dzialanie=='+') return (InA+InB);
if(Dzialanie=='-') return (InA-InB);
if(Dzialanie=='*') return (InA*InB);
if(Dzialanie=='/' && InB!=0.0) return (InA/InB);
else return 0.0;
}
```

Rozwiązanie 2

```
float DzialanieArytmetyczne(float InA, float InB, char Dzialanie)
{
if(Dzialanie=='+') return (InA+InB);
if(Dzialanie=='-') return (InA-InB);
if(Dzialanie=='*') return (InA*InB);
if(Dzialanie=='/') return (InA/InB);
}
```

Rozwiązanie 3

```
float DzialanieArytmetyczne(float InA, float InB, char Dzialanie)
{
if(Dzialanie=='+') return (InA+InB);
if(Dzialanie=='-') return (InA-InB);
if(Dzialanie=='*') return (InA*InB);
if(Dzialanie=='/' && InB!=0.0) return (InA/InB);
else return (float)0.0;
}
```

III-5. Funkcja udoskonalona o sygnalizację błędów

W zadaniu tym należy udoskonalić funkcję z poprzedniego rozdziału (*"Uniwersalna funkcja realizująca podstawowe operacje matematyczne")* o wykrywanie błędu dzielenia przez 0.

Wynik działania funkcji nie będzie już rezultatem operacji matematycznej, a statusem błędu.

Wynik równy jeden określa poprawne wykonanie operacji matematycznej, zero określa błąd (dzielenie przez 0). Ponieważ funkcja może zwracać tylko jedną wartość, to rezultat określonej operacji matematycznej będzie zwracany w postaci dodatkowego argumentu funkcji.

Należy pamiętać o tym, że jeśli istnieje konieczność zwrócenia wartości w postaci jednego z argumentów, to musi on reprezentować adres zmiennej, do której wartość zostaje zwrócona *(zob. "Funkcje"), (zob. "Wskaźniki do zmiennych")*.

Proponowane rozwiązanie:

```
BOOL DzialanieArytmetyczne(float InA, float InB,char Dzialanie, float *Wynik)
{
//------------------------------------ dodawanie --------------------------------
if(Dzialanie=='+')
{
*Wynik=(InA+InB);
return 1;
}
//------------------------------------ odejmowanie ------------------------------
if(Dzialanie=='-')
{
*Wynik=(InA-InB);
return 1;
}
//------------------------------------ mnożenie ---------------------------------
if(Dzialanie=='*')
{
*Wynik=(InA*InB);
return 1;
}
//------------------------------------ dzielenie --------------------------------
if(Dzialanie=='/' && InB!=0.0)
{
*Wynik=(InA/InB);
return 1;
}
else return 0;
}
```

Zadanie

Przykładowe wykorzystanie przedstawionej funkcji we własnym programie (wywołanie funkcji) powinno być zadeklarowane w sposób następujący (prawidłowa odpowiedź znajduje się na końcu podręcznika):

Rozwiązanie 1

```
float zm1;
if(DzialanieArytmetyczne(12,0,'/',&zm1)) printf("%f\n",zm1);
else printf("Blad operacji\n");
```

Rozwiązanie 2

```
float *zm1;
if(DzialanieArytmetyczne(12,0,'/',&zm1)) printf("%f\n",zm1);
else printf("Blad operacji\n");
```

Rozwiązanie 3

```
float zm1;
if(DzialanieArytmetyczne(12,0,'/',*zm1)) printf("%f\n",zm1);
else printf("Blad operacji\n");
```

III-6. Zamiana wartości zmiennej typu *float* na łańcuch znakowy

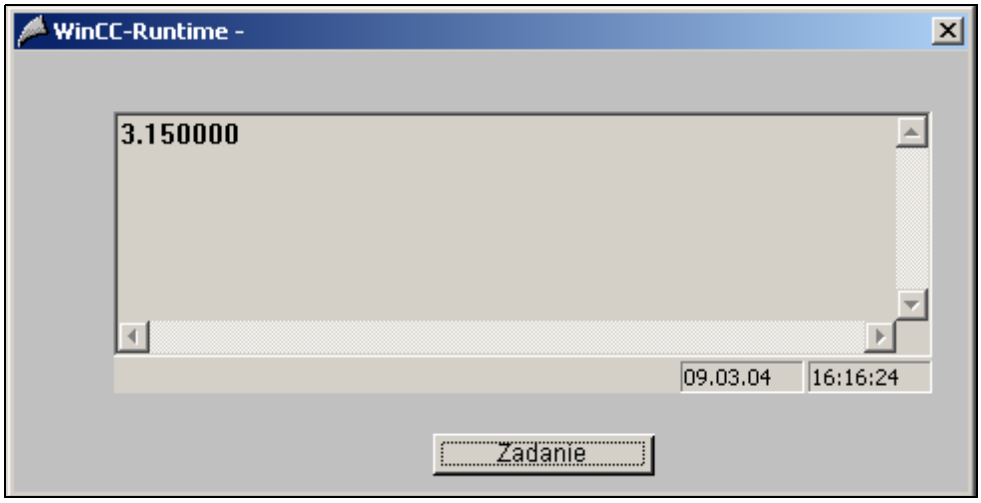
Zadanie to będzie polegało na realizacji zamiany aktualnej wartości zmiennej typu *float* na łańcuch tekstowy *(zob. "Zmienne")*.

Do tego celu wykorzystana zostanie funkcja analogiczna do funkcji *printf()*, ale wysyłająca tekst nie na ekran komputera a do zmiennej stanowiącej tablicę znakową. Mowa tutaj o funkcji *sprintf() (zob. "Wybrane funkcje biblioteczne")*.

Proponowane rozwiązanie:

```
char tekst[256];      //deklaracja łańcucha tekstowego - tablicy znaków o rozmiarze 255
                      //znaków
float zm;             //deklaracja zmiennej typu float

zm=3.15;              //wpisanie wartości do zmiennej typu float

sprintf(tekst,"%f",zm);  //zamiana wartości zmiennej typu float na tekst i wpisanie go do
                         //tablicy znaków

printf(tekst);        //wysłanie tekstu na ekran
```

W wyniku działania funkcji na ekranie zostanie wyświetlona wartość zmiennej zm:



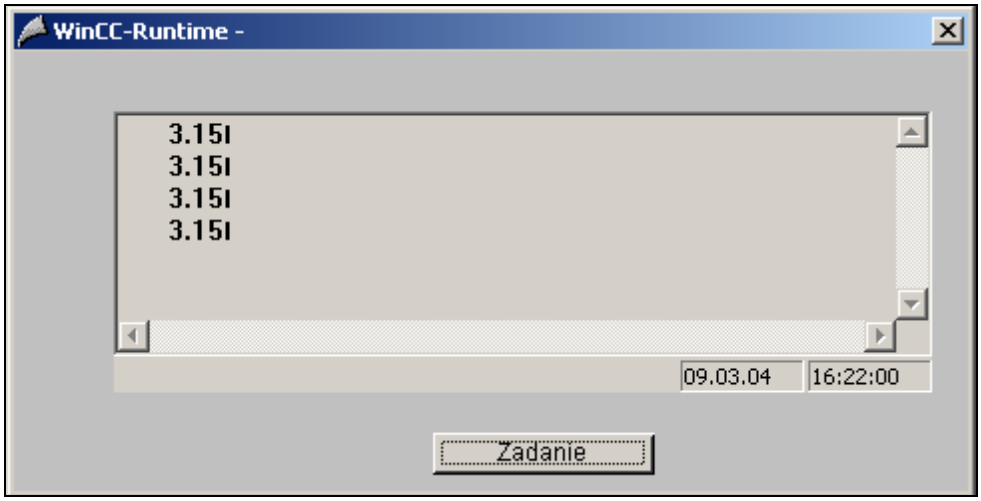
Zadanie

Powyższy program zmodyfikowano w następujący sposób:

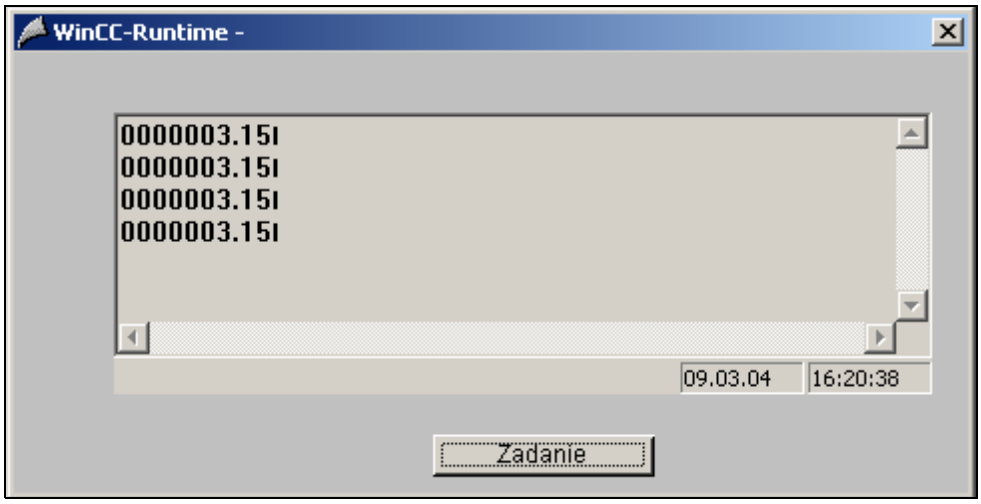
```
char tekst[256];
float zm;
zm=3.15;
sprintf(tekst,"%010.2f",zm);
printf("%s\n",tekst);
```

Kilkakrotne naciśnięcie przycisku spowoduje wyświetlenie napisu (prawidłowa odpowiedź znajduje się na końcu podręcznika):

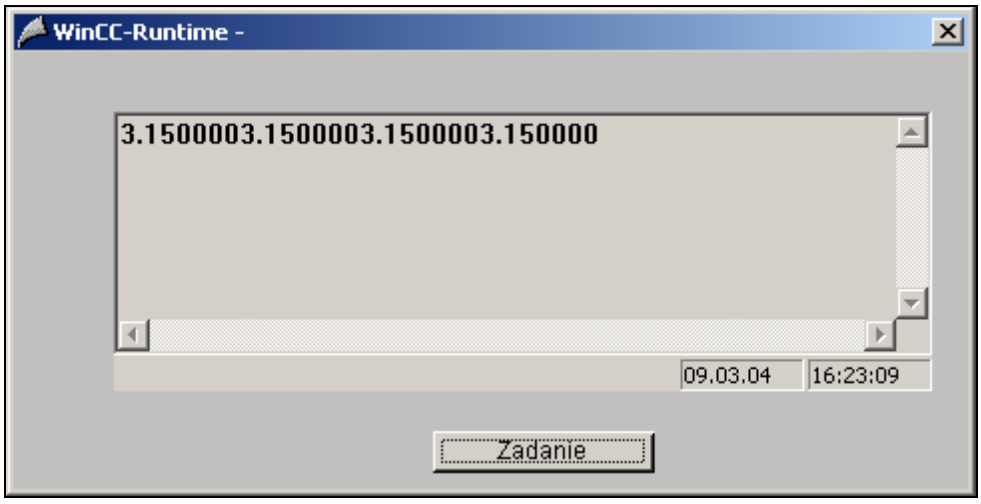
Rozwiązanie 1



Rozwiązanie 2



Rozwiązanie 3



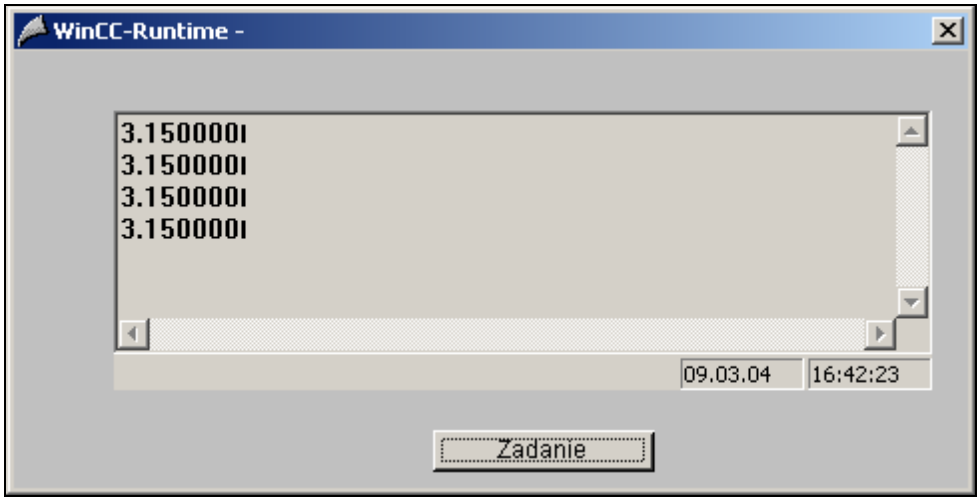
III-7. Zamiana łańcucha znakowego na zmienną typu *float*

W zadaniu tym należy zamienić aktualną wartość łańcucha tekstowego na zmienną typu *float (zob. "Zmienne")*.
Oczywiście tekst musi zawierać prawidłową wartość odpowiadającą wartości zmiennej *float*.
Do tego celu wykorzystana zostanie funkcja o działaniu odwrotnym do funkcji *sprintf()* - funkcja *sscanf() (zob. "Wybrane funkcje biblioteczne")*.

Proponowane rozwiązanie:

```
char tekst[256];      //deklaracja łańcucha tekstowego - tablicy znaków o rozmiarze 255
                      //znaków
float zm;             //deklaracja zmiennej typu float


sprintf(tekst,"%f",3.15); //wpisanie wartości do zmiennej typu tablica tekstów - funkcja
                          //sprintf() zob. "Funkcje biblioteczne")

sscanf(tekst,"%f",&zm);    //zamiana wartości zmiennej typu tekst na typ float
printf("%f\n",zm);         //wysłanie wartości zmiennej na ekran
```

W wyniku działania funkcji na ekranie zostanie wyświetlona wartość zmiennej zm:



Zadanie

Czy w wywołaniu funkcji *sprintf()* w powyższym zadaniu argument (zmienna) tekst jest traktowana jako wskaźnik (czy do wywołania funkcji *sprintf()* przekazywany jest adres zmiennej tekst, czy jedynie jej wartość) *(zob. "Funkcje"), (zob. "Wskaźniki do zmiennych")* ? Prawidłowa odpowiedź znajduje się na końcu podręcznika.

III-8. Pobranie długości tekstu zapisanego w łańcuchu znakowym

Zadanie polega na sprawdzeniu długości tekstu zapisanego w łańcuchu tekstowym. Najprostszą metodą jest wykorzystanie funkcji bibliotecznej *(zob. "Wybrane funkcje biblioteczne") - strlen()*.
Inną metodą jest wykorzystanie właściwości łańcuchów tekstowych - informacji o końcu tekstu jaki stanowi znak '\0' *(zob. "Zmienne w jezyku C").* Jeśli tablica znaków zawiera przykładowy napis *"ala"*, to kolejne elementy tablicy będą zawierać znaki: 'a', 'l', 'a', '\0', pozostałe miejsce w tablicy, która jest zazwyczaj znacznie dłuższa pozostaje niewykorzystane.

Zadanie

Które z poniższych rozwiązań jest poprawne (nie chodzi o to które jest optymalne, a jedynie o to, które będzie poprawnie działać) ? Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
int dlugosc;
char tekst[256];
sprintf(tekst,"%s"," ala ma kota");
dlugosc=strlen(tekst);
printf("dlugosc tekstu = %d\n",dlugosc);
```

Rozwiązanie 2

```
int i,j,dlugosc;
char tekst[256];
sprintf(tekst,"%s"," ala ma kota");
j=0;
for(i=0;i<=256;i++)
{
if(tekst[i]!='\0') j=i+1;
else break;
}
dlugosc=j;
printf("dlugosc tekstu = %d\n",dlugosc);
```

Rozwiązanie 3

```
int i,j,dlugosc;
char tekst[256];
sprintf(tekst,"%s"," ala ma kota");
i=0;
j=0;
while(tekst[i]!='\0')
{
j=i+1;
if(i>=256) break;
i++;
}
dlugosc=j;
printf("dlugosc tekstu = %d\n",dlugosc);
```

III-9. Zamiana łańcucha znakowego na zmienną typu *float* ze sprawdzeniem poprawności konwersji

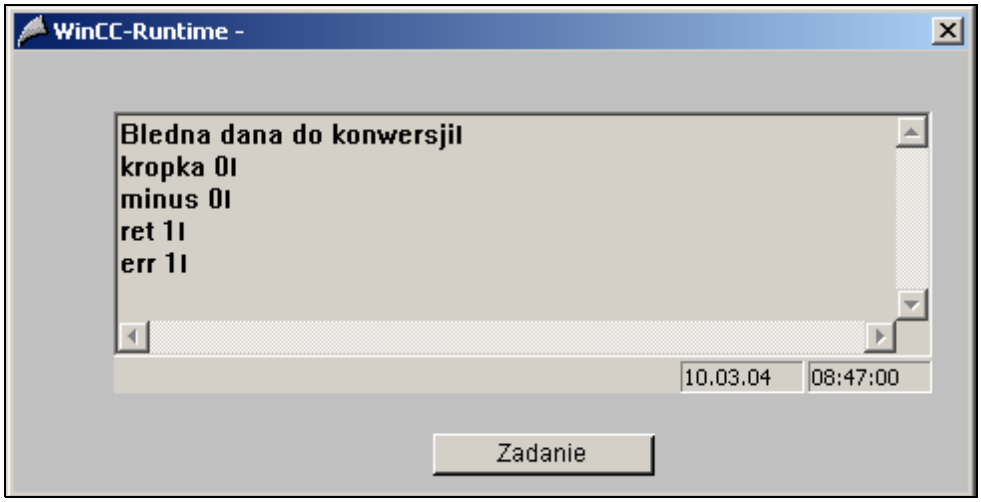
Zadanie do zrealizowania jest podobne do wykonywanego już poprzednio zadania *(zob. "Zamiana łańcucha znakowego na zmienną typu float")*.
Rozwiązanie należy uzupełnić o wykrywanie poprawności konwersji - określenie, czy tekst zapisany w łańcuchu znakowym jest odpowiednikiem poprawnej wartości typu *float*.

Proponowane rozwiązanie:

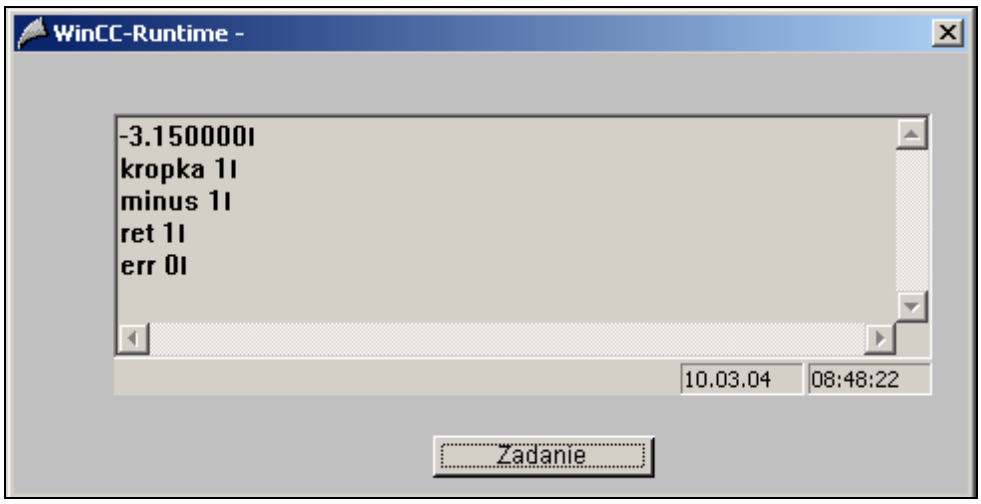
```
char tekst[256];
float zm;
int ret,err,kropka,minus;
int dlugosc;
int i;

//zadawanie różnych wartości do tablicy tekstów celem sprawdzenia działania
//sprintf(tekst,"%s","3..15");
//sprintf(tekst,"%s","3.15");
//sprintf(tekst,"%s","-003.15");
//sprintf(tekst,"%s","315s");
//sprintf(tekst,"%s","3..15");

dlugosc=strlen(tekst);
err=0;
kropka=0;
minus=0;
for(i=0;i<dlugosc;i++)
{
  if(tekst[i]>='0' && tekst[i]<='9'){;}   //czy mamy do czynienia z cyframi ?
  else
  {
   //jeśli nie są to cyfry to może być przecinek - liczenie liczby przecinków
      if(tekst[i]=='.') kropka++;
      else
      {
      //jeśli nie są to cyfry to może być minus - liczenie liczby minusów
         if(tekst[i]=='-') minus++;
         else err++;
       }
  }
}

if(kropka >1) err=1;                  //jeśli więcej niż jeden przecinek to błąd
if(minus >1) err=1;                   //jeśli więcej niż jeden minus to błąd
ret=sscanf(tekst,"%f",&zm);           //sprawdzenie poprawności konwersji funkcji sscanf()
if(ret != 1) err=1;

if(err==0) printf("%f\n",zm);         //jeśli nie ma błędu konwersji, to wyświetlamy wartość
                                      //zmiennej float
else printf("Bledna dana do konwersji\n");       //w innym przypadku komunikat o błędzie
```

```
//wyświetlenie wartości zmiennych pomocniczych do celów diagnostycznych

printf("kropka %d\n",kropka);
printf("minus %d\n",minus);
printf("ret %d\n",ret);
printf("err %d\n",err);
```

W wyniku działania funkcji na ekranie zostaną wyświetlone następujące teksty



lub w przypadku poprawnej konwersji



Zadanie

Który z poniższych programów nie zawiera błędów składni języka C (prawidłowa odpowiedź znajduje się na końcu podręcznika):

Rozwiązanie 1

```
int i,dlugosc,j;
dlugosc=15;
for(i=0;i<dlugosc;i++)
{
  if(j==5) printf("ala ma kota\n");
  j=i;
}
```

Rozwiązanie 2

```
int i,dlugosc,j;
dlugosc=15;
for(i=0,i<dlugosc,i++)
{
  if(j==5) printf("ala ma kota\n");
  j=i;
}
```

Rozwiązanie 3

```
int i,dlugosc,j;
dlugosc:=15;
for(i==0;i<dlugosc;i++)
{
  if(j=5) printf("ala ma kota\n");
  j=i;
}
```

III-10. Skalowanie wartości z określeniem progów przekroczeń

Zadanie polega na przeskalowaniu wartości typu *float* oraz określeniu przekroczeń tej wartości (przekroczenia określane są dla wartości już przeskalowanej).

Proponowane rozwiązanie:

```
//deklaracje odpowiednich zmiennych
float X_Lo;
float X_Hi;
float Y_Lo;
float Y_Hi;
float InVal;
float Y_LimitLo;
float Y_LimitHi;
float OutVal;
BOOL OutHiLimit;
BOOL OutLoLimit;
float a,b;

InVal=11;       //tymczasowe wartości zadawane do symulacji
X_Lo=0;
X_Hi=100;
Y_Lo=0;
Y_Hi=200;
Y_LimitLo=90;
Y_LimitHi=210;

if((X_Lo-X_Hi)==0.0) return;     //błąd dzielenia przez 0

a=(Y_Lo-Y_Hi)/(X_Lo-X_Hi);   //obliczenie współczynników a i b dla równania prostej
b=Y_Lo-a*X_Lo;
OutVal=a*InVal+b;

if(OutVal>Y_LimitHi) OutHiLimit=1;     //odpowiednie ustawienie zmiennych limitów
else OutHiLimit=0;

if(OutVal<Y_LimitLo) OutLoLimit=1;
else OutLoLimit=0;

printf("OutVal = %f\n",OutVal);              //wyświetlenie informacji na ekranie
printf("OutHiLimit= %d\n",OutHiLimit);
printf("OutLoLimit= %d\n",OutLoLimit);
```

Zadanie

Na ekranie należy wyświetlić zmienną typu *float* prezentowaną w apostrofach, na 10 miejscach w tym 2 po przecinku, z przodu powinna być uzupełniona zerami. Zmienna powinna być opatrzona opisem pobranym z innej zmiennej tekstowej, powinna być wyświetlona zawsze w nowej linii. Prawidłowa odpowiedź znajduje się na końcu podręcznika.

```
Opis do zmiennej '0000256.87'
```

Które rozwiązanie jest prawidłowe?

Rozwiązanie 1

```
char txt[256];
float zm1;
sscanf(txt,"%s","Opis do zmiennej");
zm1=256.87;
printf("%s \'%10.2f\'\n",txt,zm1);
```

Rozwiązanie 2

```
char txt[256];
float zm1;
sprintf(txt,"%s","Opis do zmiennej");
zm1=256.87;
printf("%s \'%010.2f\'\n",txt,zm1);
```

Rozwiązanie 3

```
char txt[256];
float zm1;
sprintf(txt,"%s","Opis do zmiennej");
zm1=256.87;
printf("%s %f\'\n",txt,zm1);
```

III-11. Wykonanie funkcji bibliotecznej

Stworzyć funkcję biblioteczną do poprzedniego zadania *(zob. "Skalowanie wartości z określeniem progów przekroczeń")*.

Należy tutaj przypomnieć metody zwracania wartości poprzez argumenty wejściowe funkcji *(zob."Funkcje", "Wskaźniki do zmiennych").*

Proponowane rozwiązanie:

```
BOOL Skalowanie( float X_Lo,
                 float X_Hi,
                 float Y_Lo,
                 float Y_Hi,
                 float InVal,
                 float Y_LimitLo,
                 float Y_LimitHi,
                 float *OutVal,
                 BOOL *OutHiLimit,
                 BOOL *OutLoLimit)
{
float a,b;

if((X_Lo-X_Hi)==0.0) return 0;      //błąd dzielenia przez 0

//obliczenie współczynników a i b dla równania prostej
a=(Y_Lo-Y_Hi)/(X_Lo-X_Hi); b=Y_Lo-a*X_Lo;
*OutVal=a*InVal+b;

  //odpowiednie ustawienie zmiennych limitów
  if (*OutVal>Y_LimitHi)     *OutHiLimit=1;
  else *OutHiLimit=0;

  if (*OutVal<Y_LimitLo)     *OutLoLimit=1;
  else *OutLoLimit=0;

  return 1;
}
```

Zadanie

Jak powinna być wywołana powyższa funkcja biblioteczna, jeśli chcemy jej użyć we własnym programie? Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
float InVal;
float &OutVal;
BOOL &OutHiLimit;
BOOL &OutLoLimit;
InVal=256.8;

if(Skalowanie(0,100,0,200,InVal,90,210,OutVal,OutHiLimit,OutLoLimit))
{
printf("OutVal = %f\n",OutVal);
printf("OutHiLimit= %d\n",OutHiLimit);
printf("OutLoLimit= %d\n",OutLoLimit);
}
else printf("Dzielenie przez zero\n");
```

Rozwiązanie 2

```
float InVal;
float *OutVal;
BOOL *OutHiLimit;
BOOL *OutLoLimit;
InVal=256.8;

if(Skalowanie(0,100,0,200,InVal,90,210,OutVal,OutHiLimit,OutLoLimit))
{
printf("OutVal = %f\n",OutVal);
printf("OutHiLimit= %d\n",OutHiLimit);
printf("OutLoLimit= %d\n",OutLoLimit);
}
else printf("Dzielenie przez zero\n");
```

Rozwiązanie 3

```
float InVal;
float OutVal;
BOOL OutHiLimit;
BOOL OutLoLimit;
InVal=256.8;

if(Skalowanie(0,100,0,200,InVal,90,210,&OutVal,&OutHiLimit,&OutLoLimit))
{
printf("OutVal = %f\n",OutVal);
printf("OutHiLimit= %d\n",OutHiLimit);
printf("OutLoLimit= %d\n",OutLoLimit);
}
else printf("Dzielenie przez zero\n");
```

III-12. Porównywanie dwóch łańcuchów znakowych

Zapisać funkcję porównującą zawartość dwu łańcuchów znakowych. Jeśli są identyczne, zwrócona powinna zostać wartość 1, w przeciwnym wypadku funkcja powinna zwracać wartość 0.

W rozwiązaniu nie należy korzystać z innych funkcji bibliotecznych języka C niż *strlen() (zob. "Wybrane funkcje biblioteczne")*

Proponowane rozwiązanie:

```
BOOL  PorownajTeksty(char *txt1,char *txt2)
{
  int dlugosc1,dlugosc2,i;

  dlugosc1=strlen(txt1);
  dlugosc2=strlen(txt2);                    //pobierz długości obu łańcuchów znakowych

  //jeśli łańcuchy mają inne długości to nie mogą być takie same - zwróć 0
  if(dlugosc1 != dlugosc2) return 0;

  for(i=0;i<=dlugosc1;i++)

  //sprawdź poszczególne znaki w łańcuchach - jeśli są różne zwróć 0
  if(*(txt1+i)!=*(txt2+i)) return 0;

  //oba łańcuchy są identyczne - zwróć 1
  return 1;
}
```

Zadanie

Które wywołania funkcji z powyższego zadania są prawidłowe ?
Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
char t1[]="ola ma kota";
char t2[256];
sprintf(t2,"%s","ala ma kota");
printf("%d\n",PorownajTeksty(t1,t2));
```

Rozwiązanie 2

```
char *t1="ala ma kota";
char t2[256];
sprintf(t2,"%s","ala ma kota");
printf("%d\n",PorownajTeksty(t1,t2));
```

Rozwiązanie 3

```
char t1[256];
char t2[256];
sprintf(t1,"%s","ala ma kota ");
sprintf(t2,"%s","ala ma kota");
printf("%d\n",PorownajTeksty(t1,t2));
```

III-13. Operacje na bitach

Zapisać funkcję sprawdzającą stan bitu w zmiennej typu słowo (16 bitowej).
Funkcja powinna posiadać dwa argumenty wejściowe: *wartosc* - zmienna typu *WORD*, w której sprawdzane będą wartości poszczególnych bitów, *nr_bitu* - numer bitu którego stan chcemy sprawdzić (0..15).
Wynikiem działania funkcji powinien być stan sprawdzanego bitu.

Proponowane rozwiązanie:

```
BOOL ZwrocWartoscBituONr(WORD wartosc,WORD nr_bitu)
{
 WORD wart_bitu,i;
 BOOL Out;

 Out=0;
 wart_bitu=1;
 i=0;

  //wygenerowanie odpowiedniej maski bitowej zależnej od numeru
  while(i<nr_bitu)
  {
   //obserwowanego bitu - celem późniejszego sprawdzenia stanu tego bitu
   wart_bitu=(WORD)(wart_bitu*2);
   i++;
  }

 wartosc= (WORD)(wartosc &wart_bitu);        //sprawdzenie czy dany bit w słowie jest
                                             //jedynką
 if(wartosc==wart_bitu) Out=1;

 return Out;

}
```

Zadanie

Które wywołania funkcji z powyższego zadania dadzą w wyniku wartość 1 ?
Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
WORD ObserwowaneSlowo;
WORD NrBitu;
BOOL ret;
ObserwowaneSlowo=0x01;
NrBitu=0;
ret=ZwrocWartoscBituONr(ObserwowaneSlowo,NrBitu);
printf("Bit nr %d ma wartosc %d",NrBitu,ret);
```

Rozwiązanie 2

```
WORD ObserwowaneSlowo;
WORD NrBitu;
BOOL ret;
ObserwowaneSlowo=100;
NrBitu=2;
ret=ZwrocWartoscBituONr(ObserwowaneSlowo,NrBitu);
printf("Bit nr %d ma wartosc %d",NrBitu,ret);
```

Rozwiązanie 3

```
WORD ObserwowaneSlowo;
WORD NrBitu;
BOOL ret;
ObserwowaneSlowo=0x64;
NrBitu=5;
ret=ZwrocWartoscBituONr(ObserwowaneSlowo,NrBitu);
printf("Bit nr %d ma wartosc %d",NrBitu,ret);
```

III-14. Zapis do pliku tekstowego

Stworzyć funkcję wpisującą do pliku tekstowego o podanej nazwie określony tekst *(zob. "Wybrane funkcje biblioteczne")*.

Jeśli plik jeszcze nie istnieje, to funkcja powinna go utworzyć, w przeciwnym wypadku powinna dopisywać teksty do pliku już istniejącego.

Funkcja powinna posiadać dwa argumenty wejściowe: *nazwa_pliku* - zmienna typu *char**, w której przekazywana będzie nazwa pliku do obróbki, *tekst* - tekst do zapisu w pliku. Wynikiem działania funkcji powinien być status operacji na pliku - 1 operacja się udała, 0 - błąd zapisu. Proponowane rozwiązanie:

```
BOOL ZapiszTekstDoPliku(char *nazwa_pliku, char *tekst)
{

  FILE *pFile;                    //deklaracja zmiennej - uchwytu pliku

  pFile=fopen(nazwa_pliku,"at");  //odpowiednie otwarcie pliku
  if(pFile)
  {

   fprintf(pFile,tekst);          //zapisz tekst do pliku

   fclose(pFile);                 //zamknięcie pliku

   return 1;
  }
 else return 0;

}
```

Zadanie

Które wywołanie funkcji z powyższego zadania jest prawidłowe ?
Prawidłowa odpowiedź znajduje się na końcu podręcznika.

Rozwiązanie 1

```
char NazwaPliku[]="c://plik1.txt";
ZapiszTekstDoPliku(NazwaPliku,"ala ma kota\n");
```

Rozwiązanie 2

```
ZapiszTekstDoPliku("c:\plik1.txt","ala ma kota\n");
```

Rozwiązanie 3

```
char *NazwaPliku="c:\\plik1.txt";
ZapiszTekstDoPliku(NazwaPliku,"ala ma kota\n");
```

IV. Rozwiązania zadań

IV-1. Mój pierwszy program

Rozwiązanie 1

Poprawne rozwiązanie.

Rozwiązanie 2

Złe rozwiązanie.
Funkcje printf() w prezentowanym przykładzie nie posiadają odwołań do odpowiednich zmiennych. Zapis prawidłowy to np.: printf("zmienna1 = %.2f \n", zm1);

Rozwiązanie 3

Złe rozwiązanie.
Funkcje printf() nie posiadają znacznika rozmiaru formatowanej zmiennej.
Nie wiadomo więc na ilu miejscach zmienne te powinny być wyświetlane.

IV-2. Podstawowe operacje matematyczne

Rozwiązanie 1

Złe rozwiązanie.
Ten kod programu nie stanowi funkcji.

Rozwiązanie 2

Złe rozwiązanie.
Funkcja z zadeklarowanym typem zwracanym - void nie może zwracać wartości.

Rozwiązanie 3

Poprawne rozwiązanie.

IV-3. Wykorzystanie instrukcji if()

Rozwiązanie

255.
Realizowana operacja matematyczna to:
zm3=(15+10)*10=250 po instrukcji else: zm3=250+5

IV-4. Uniwersalna funkcja realizująca podstawowe operacje matematyczne

Rozwiązanie 1

Złe rozwiązanie.
Funkcja z zadeklarowanym typem zwracanym - void nie może zwracać wartości.

Rozwiązanie 2

Złe rozwiązanie.
Brak sprawdzania błędu dzielenia przez 0.

Rozwiązanie 3

Poprawne rozwiązanie.

IV-5. Funkcja udoskonalona o sygnalizację błędów

Rozwiązanie 1

Poprawne rozwiązanie.

Rozwiązanie 2

Złe rozwiązanie.
Źle zadeklarowana zmienna - argument funkcji.

Rozwiązanie 3

Złe rozwiązanie.
Złe odwołanie do wskaźnika zmiennej w wywołaniu funkcji.

IV-6. Zamiana wartości zmiennej typu float na łańcuch znakowy

Rozwiązanie 1

Złe rozwiązanie.
Wyświetlana zmienna będzie uzupełniona z przodu zerami.

Rozwiązanie 2

Poprawne rozwiązanie.

Rozwiązanie 3

Złe rozwiązanie.
Brakuje przejść do następnej linii.

IV-7. Zamiana łańcucha znakowego na zmienną typu float

Rozwiązanie:

Przekazywany jest adres zmiennej *text*.

Ponieważ do tej zmiennej funkcja zapisuje wynik swojego działania -
to w wywołaniu musi się znaleźć adres zmiennej - wskaźnik do zmiennej.

IV-8. Pobranie długości tekstu zapisanego w łańcuchu znakowym

Wszystkie rozwiązania są poprawne.

IV-9. Zamiana łańcucha znakowego na zmienną typu float ze sprawdzeniem poprawności konwersji

Rozwiązanie 1

Poprawne rozwiązanie.

Rozwiązanie 3

Złe rozwiązanie. W funkcji for() należy używać ';' a nie ','.

Rozwiązanie 3

Złe rozwiązanie.
W przypisaniu wartości do zmiennej wykorzystuje się znak '=', a nie znaki ':='.
W pętli for() nie używa się znaków '=='

IV-10. Skalowanie wartości z określeniem progów przekroczeń

Rozwiązanie 1

Złe rozwiązanie.
Brak uzupełniania zerami zmiennej formatowanej.

Rozwiązanie 2

Poprawne rozwiązanie.

Rozwiązanie 3

Złe rozwiązanie.
Brak określenia rozmiaru prezentowanej zmiennej.

IV-11. Wykonanie funkcji bibliotecznej

Rozwiązanie 1

Złe rozwiązanie.
Złe deklaracje zmiennych i późniejsze złe ich wywołanie.

Rozwiązanie 2

Złe rozwiązanie.
Złe deklaracje zmiennych i późniejsze złe ich wywołanie.

Rozwiązanie 3

Poprawne rozwiązanie.

IV-12. Porównywanie dwóch łańcuchów znakowych

Wszystkie rozwiązania są poprawne.

IV-13. Operacje na bitach

Wszystkie rozwiązania są poprawne.

IV-14. Zapis do pliku tekstowego

Rozwiązanie 1

Złe rozwiązanie.
Źle zadeklarowana nazwa pliku - zamiast znaków '//' w notacji języka C powinno być '\\'.

Rozwiązanie 2

Złe rozwiązanie.
Źle zadeklarowana nazwa pliku - zamiast znaku '\' w notacji języka C powinno być '\\'.

Rozwiązanie 3

Poprawne rozwiązanie.