Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 2

Kod przedmiotu: **EZ1E3012**

(studia niestacjonarne)

JĘZYK C - STRUKTURY, POLA BITOWE, UNIE

Numer ćwiczenia

INF23Z

Autor:
dr inż. Jarosław Forenc

Białystok 2017

Spis treści

Materiały dydaktyczne przeznaczone dla studentów Wydziału Elektrycznego PB.

1. Opis stanowiska

1.1. Stosowana aparatura

Podczas zajęć wykorzystywany jest komputer klasy PC z systemem operacyjnym Microsoft Windows (XP/7/10).

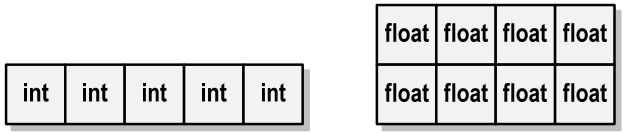
1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Microsoft Visual Studio 2008 Standard Edition lub Microsoft Visual Studio 2008 Express Edition zawierające kompilator Microsoft Visual C++ 2008.

2. Wiadomości teoretyczne

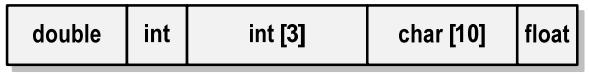
2.1. Struktury

W języku C do przechowywania wielu elementów tego samego typu stosowane są tablice jedno- lub wielowymiarowe (Rys. 1).



Rys. 1. Reprezentacja tablicy jednowymiarowej (wektora) i tablicy dwuwymiarowej (macierzy) w języku C

W przypadku, gdy pod jedną nazwą mają być zgrupowane elementy różnych typów, stosowane są **struktury**. Struktury służą zatem do reprezentacji złożonych obiektów różnych danych (Rys. 2).



Rys. 2. Reprezentacja struktury w języku C

Ogólna postać deklaracji struktury jest następująca:

```
struct nazwa_struktury
{
    typ nazwa_pola_1;
    typ nazwa_pola_2;
    ...
    typ nazwa_pola_n;
};
```

Deklaracja struktury rozpoczyna się od słowa kluczowego **struct**, po którym może występować opcjonalna **nazwa struktury** (**etykieta**). Pomiędzy nawiasami klamrowymi umieszczone są **pola struktury** (**dane**, **komponenty**, **składowe**), mające taką samą postać jak deklaracje zmiennych w programie. Deklaracja struktury kończy się średnikiem umieszczonym po nawiasie klamrowym zamykającym. Brak tego średnika jest najczęściej popełnianym błędem przy deklaracji struktury.

Poniżej przedstawione są przykłady deklaracji dwóch struktur. Struktura **punkt** przechowuje współrzędne **x** i **y** punktu w prostokątnym układzie współrzędnych. Struktura **osoba** opisuje dane osobowe: **imię** i **nazwisko** w postaci tablic znaków oraz **wiek**, **wzrost** i **wagę** jako wartości całkowite.

```
struct punkt
{
    float x;
    float y;
};
```

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int wiek;
    int wzrost, waga;
};
```

Przy nazywaniu pól struktury obowiązują takie same zasady jak przy nadawaniu nazw innym identyfikatorom. Nazwy pól struktury mogą być takie same jak nazwy innych zmiennych w programie, a nawet takie same jak nazwa struktury. Pola jednego typu mogą być umieszczone w jednym wierszu.

Deklarując strukturę wprowadzamy **nowy typ danych** (np. **struct punkt**), którym posługujemy się tak samo jak każdym innym typem standardowym. Deklaracja struktury nie tworzy obiektu (nie przydziela pamięci na pola), ale tylko

określa z czego się on składa. Aby do struktury można było zapisać dane należy zdefiniować **zmienną strukturalną**. Definicja taka może być połączona z deklaracją struktury. W takim przypadku nazwy zmiennych strukturalnych umieszcza się bezpośrednio po klamrze kończącej listę pól struktury.

```
struct osoba
{
    char imie[15];
    char nazwisko[20];
    int  wiek, wzrost, waga;
} Kowalski, Nowak;
```

W powyższej deklaracji **Kowalski** i **Nowak** są zmiennymi typu **struct osoba**. Zmienna **Kowalski** może więc przechowywać imię, nazwisko, wiek, wzrost i wagę jednej osoby. Zmienna **Nowak** może przechowywać dane takiego samego typu, ale należące już do innej osoby (Rys. 3).

Rys. 3. Zmienne strukturalne **Kowalski** i **Nowak**

Deklaracja struktury może znajdować się w dowolnym miejscu programu. Jednakże należy pamiętać o tym, że umieszczenie jej wewnątrz definicji funkcji spowoduje ograniczenie zakresu jej widzialności tylko do tej funkcji. W poniższym programie deklaracja struktury **osoba** znajduje się przed funkcją **main()**. Jest to sytuacja najczęściej spotykana w praktyce. Przykład ten pokazuje także inny sposób deklaracji zmiennych strukturalnych. Bezpośrednio po klamrze kończącej listę pól struktury znajduje się średnik, natomiast deklaracje zmiennych strukturalnych zostały umieszczone na początku funkcji **main()**. Deklaracje te mają

taką samą postać jak deklaracje zmiennych standardowych typów (**struct osoba** - nazwa typu, **Kowalski**, **Nowak** - nazwy zmiennych).

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int  wiek, wzrost, waga;
};

int main(void)
{
    struct osoba Kowalski;
    struct osoba Nowak;

    /* ... */

    return 0;
}
```

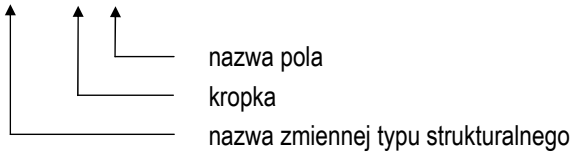
2.2. Odwołania do pól struktury

Dostęp do pól struktury możliwy jest dzięki konstrukcji typu:

```
nazwa_struktury.nazwa_pola
```

W poniższym przykładzie zapisujemy wartość **25** do pola **wiek** zmiennej strukturalnej **Kowalski**.

```
Kowalski.wiek = 25;
```



Operator kropki nazywany jest operatorem bezpośredniego wyboru pola. Z wyrażenia **Kowalski.wiek** można korzystać dokładnie w ten sam sposób jak z każdej innej zmiennej typu **int**.

Zapisanie wartości **"Jan"** do pola **imie** (tablicy znaków) wymaga zastosowania funkcji **strcpy()**.

```
strcpy(Kowalski.imie,"Jan");
```

W przypadku posługiwania się **wskaźnikiem** do zmiennej strukturalnej odwołania do pól struktury wymagają użycia operatora pośredniego wyboru pola zapisywanego w postaci znaku minus i znaku większości (**->**), np.

```
struct osoba Kowalski, *PtrKowalski;
PtrKowalski = &Kowalski;
PtrKowalski -> wiek = 25;
strcpy(PtrKowalski -> imie,"Jan");
```

lub

```
(*PtrKowalski).wiek = 25;
strcpy((*PtrKowalski).imie,"Jan");
```

W ostatnim zapisie dodatkowe nawiasy są konieczne, gdyż operator **.** ma wyższy priorytet niż operator *.

Zapisanie danych do zmiennej strukturalnej i wyświetlenie ich na ekranie.

```
#include <stdio.h>

struct osoba
{
    char imie[15];
    char nazwisko[20];
    int  wiek;
};

int main(void)
{
    struct osoba Nowak;
    printf("Imie:     "); scanf("%s",Nowak.imie);
    printf("Nazwisko: "); scanf("%s",Nowak.nazwisko);
    printf("Wiek:     "); scanf("%d",&Nowak.wiek);

    printf("%s %s, wiek: %d\n",Nowak.imie,
                       Nowak.nazwisko,Nowak.wiek);

    return 0;
}
```

Przykładowy wynik uruchomienia programu:

```
Imie:     Jan
Nazwisko: Nowak
Wiek:     25
Jan Nowak, wiek: 25
```

2.3. Inicjalizacja zmiennej strukturalnej

W deklaracjach pól struktury nie mogą występować inicjalizacje. Można natomiast inicjalizować zmienne strukturalne, np.

```
struct osoba Nowak = {"Jan","Nowak",25};
```

Brakujące pola w inicjalizacji są zastępowane zerami. Każda wartość początkowa musi być wyrażeniem stałym (nie może być zmienną).

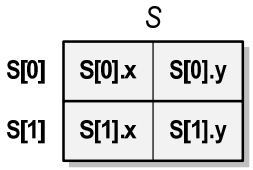
Zmienne strukturalne tego samego typu można sobie przypisywać (nawet jeśli zawierają tablice znaków), ale nie można ich porównywać ze sobą.

```
struct osoba Nowak = {"Jan","Nowak",25};
struct osoba Nowak1;
Nowak1 = Nowak;
```

2.4. Złożone deklaracje struktur

W języku C można deklarować tablice struktur. Poniższy przykład zawiera deklarację 2-elementowej tablicy **S** (Rys. 4) opisującej położenie odcinka w prostokątnym układzie współrzędnych (współrzędne jego początku i końca).

```
struct punkt
{
    float x;
    float y;
} S[2];
```



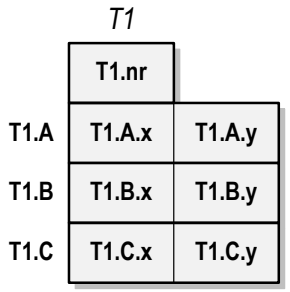
Rys. 4. Tablica **S** zawierająca dwie struktury **punkt**

Odwołując się do poszczególnych pól tablicy struktur należy podać nazwę tablicy, indeks jej elementu (w nawiasach kwadratowych) oraz (po kropce) nazwę pola struktury.

```
S[0].x = 1;
S[0].y = 2;

S[1].x = 5;
S[1].y = 3;
```

Jako pole struktury może występować inna struktura (jest to tzw. zagnieżdżanie struktur). W poniższym przykładzie struktura **trojkat** (Rys. 5) zawiera cztery pola, z czego trzy są strukturami **punkt**.



Rys. 5. Zmienna strukturalna **T1** zawierająca cztery pola

Obliczenie pola trójkąta na podstawie współrzędnych jego wierzchołków.

```
#include <stdio.h>
#include <math.h>

struct punkt
{
    float x;
    float y;
};

struct trojkat
{
    int nr;
    struct punkt A;
    struct punkt B;
    struct punkt C;
};

int main(void)
{
    struct trojkat T1;
    float pole;

    T1.nr = 1;

    T1.A.x = 1;
    T1.A.y = 2;

    T1.B.x = 5;
    T1.B.y = 3;
```

```
    T1.C.x = 2;
    T1.C.y = 4;

    pole = fabs(T1.A.x*T1.B.y + T1.B.x*T1.C.y
              + T1.C.x*T1.B.y - T1.C.x*T1.B.y
              - T1.A.x*T1.C.y - T1.B.x*T1.A.y) / 2;

    printf("A(%g,%g)\n",T1.A.x,T1.A.y);
    printf("B(%g,%g)\n",T1.B.x,T1.B.y);
    printf("C(%g,%g)\n",T1.C.x,T1.C.y);
    printf("Pole trojkata = %g\n",pole);

    return 0;
}
```

Wynik działania programu:

```
A(1,2)
B(5,3)
C(2,4)
Pole trojkata = 4.5
```

W odwołaniach do pól zagnieżdżonych struktur operator kropki występuje wielokrotnie. Zapis **T1.A.x** jest interpretowany od strony lewej do prawej, a zatem odpowiada zapisowi **(T1.A).x**.

2.5. Pola bitowe

Pola bitowe umożliwiają dostęp do pojedynczych bitów oraz przechowywanie małych wartości zajmujących pojedyncze bity bez zbędnych strat pamięci. Pola bitowe deklaruje się wewnątrz struktur w następujący sposób:

```
typ id_pola : wielkość_pola
```

- wielkość_pola — obszar zajmowany przez pole, w bitach
- id_pola — opcjonalny identyfikator (nazwa) pola
- typ — (**int**, **unsigned int**, **signed int**)

Pola zajmują tyle bitów, ile podano jako **wielkość_pola**. Wartości zapisane w polach traktowane są jak liczby całkowite. Zakres wartości pól wynika z **wielkości_pola**.

```
struct Flags_8086
{
    unsigned int CF : 1;    /* Carry Flag */
    unsigned int    : 1;
    unsigned int PF : 1;    /* Parity Flag */
    unsigned int    : 1;
    unsigned int AF : 1;    /* Auxiliary - Carry Flag */
    unsigned int    : 1;
    unsigned int ZF : 1;    /* Zero Flag */
    unsigned int SF : 1;    /* Signum Flag */
    unsigned int TF : 1;    /* Trap Flag */
    unsigned int IF : 1;    /* Interrupt Flag */
    unsigned int DF : 1;    /* Direction Flag */
    unsigned int OF : 1;    /* Overflow Flag */
};

struct Bits
{
    unsigned int a : 4;
    unsigned int b : 2;
    unsigned int   : 4;
    unsigned int c : 6;
};
```

Dostęp do pól bitowych odbywa się na takiej samej zasadzie jak do normalnych pól struktury.

```
struct Bits Word;
Word.a = 10;
Word.b = 3;
```

Poniższe przypisanie jest niepoprawne, gdyż pole **b** zajmuje 2 bity, a do zapisania liczby **4** potrzebne są trzy bity ($4_{10} = 100_2$).

```
Word.b = 4;
```

Jeśli pole nie ma podanej nazwy, to nie można się do niego odwoływać. Tego typu pola stosuje się w sytuacji, gdy określone bity pomiędzy polami nie są używane.

Pola bitowe nie posiadają adresów. Wobec pola bitowego nie można więc stosować operatora pobierania adresu **&**. Konsekwencją tego jest brak możliwości nadania wartości polu bitowemu funkcją **scanf()**.

2.6. Unie

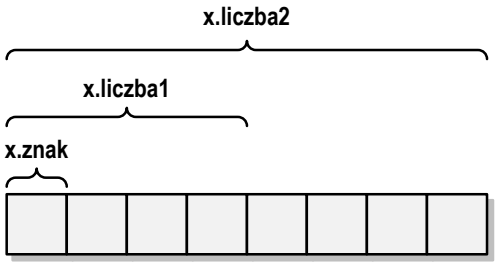
Unia jest specjalnym rodzajem struktury umożliwiającym przechowywanie danych różnych typów w tym samym obszarze pamięci. Unię deklaruje się w podobny sposób jak strukturę:

```
union zbior
{
    char   znak;
    int    liczba1;
    double liczba2;
};
```

Do przechowywania wartości w unii konieczne jest zadeklarowanie odpowiedniej zmiennej. Można deklarować pojedyncze zmienne, tablice unii i wskaźniki do unii.

```
union zbior x;
union zbior tab[5];
union zbior *ptr;
```

Zmienna **x** z powyższego przykładu może przechowywać wartość typu **char** lub wartość typu **int** lub wartość typu **double**, ale tylko jedną z nich w danym momencie (Rys. 6). Rozmiar unii wyznaczany jest przez rozmiar największego jej pola. W przypadku unii **zbior** wynosi on 8 bajtów, gdyż taki jest rozmiar pola **liczba2** (typ **double**).



Rys. 6. Interpretacja graficzna unii

Dostęp do pól unii jest taki sam jak do pól struktury, np.

```
x.znak = 'a';
x.liczba2 = 12.15;

ptr = &x;
ptr->liczba1 = -25;
```

Unię można zainicjować jedynie wartością o typie jej pierwszej składowej. Unie tego samego typu można sobie przypisywać.

```
union zbior y1 = {'a'};
union zbior y2;

y2 = y1;
```

3. Przebieg ćwiczenia

Na pracowni specjalistycznej należy wykonać wybrane zadania wskazane przez prowadzącego zajęcia. W różnych grupach mogą być wykonywane różne zadania.

1. Zdefiniuj strukturę **paszport** zawierającą następujące pola: **nazwisko**, **imię**, **obywatelstwo**, **seria**, **numer**, **płeć**. Następnie zadeklaruj dwie zmienne strukturalne. Pierwszą z nich zainicjalizuj wartościami, natomiast wartości do drugiej wczytaj w klawiatury. Wyświetl na ekranie zawartość obu zmiennych.

2. Struktura **zesp** przechowuje liczbę zespoloną.

```
struct zesp
{
    float Re, Im;
};
```

Zadeklaruj dwie zmienne typu strukturalnego (**z1**, **z2**) i zainicjalizuj je podanymi wartościami: **z1 = 2 + j4 z2 = 6 - j8**

Oblicz sumę, różnicę, iloraz oraz iloczyn liczb zespolonych **z1** i **z2**. Wyniki przypisz do zmiennej strukturalnej **z3**. Sprawdź poprawność otrzymanych wyników z tabelą 1.

Tabela 1. Poprawne wyniki do zadania 2

Operacja	Wynik
z1 + z2	8 - j4
z1 - z2	-4 + j12
z1 * z2	44 + j8
z1 / z2	-0,2 + j0,4

3. Struktura **rezonans** przechowuje informacje o szeregowym obwodzie rezonansowym: **rezystancja**, **indukcyjność**, **pojemność**, **częstotliwość rezonansowa**, **impedancja falowa**, **dobroć**. Napisz program, w którym użytkownik wprowadza parametry obwodu (**R**, **L**, **C**), a program oblicza i wyświetla pozostałe wielkości charakteryzujące stan rezonansu. Wszystkie wielkości i parametry muszą być przechowywane w strukturze. Dodatkowo program powinien sprawdzić poprawność wprowadzonych danych (**R>0**, **L>0**, **C>0**). W przypadku błędu program wyświetla odpowiedni komunikat i prosi jeszcze raz o wprowadzenie danych, itd., aż do momentu, gdy dane będą prawidłowe.

częstotliwość: $$f = \frac{1}{2\pi\sqrt{LC}} \quad (1)$$

impedancja falowa: $$\rho = \sqrt{\frac{L}{C}} \quad (2)$$

dobroć: $$Q = \frac{\rho}{R} \quad (3)$$

Przykład działania programu:

```
Podaj R: -5
Podaj L: 0.01
Podaj C: 2.5e-4
Bledne dane.
Podaj dane jeszcze raz.

Podaj R: 10
Podaj L: 0.01
Podaj C: 2.5e-4

Czestotliwosc rezonansowa: 100.65842
Impedancja falowa:         6.32455
Dobroc:                    0.63246
```

4. **N** - elementowa tablica zawiera struktury **punkt** opisujące współrzędne punktów (**x, y**) w prostokątnym układzie współrzędnych. Napisz program, który odnajdzie i wyświetli numer oraz współrzędne punktu, którego odległość od początku układu współrzędnych jest:

a) największa;
b) najmniejsza.

Współrzędne punktów wygeneruj pseudolosowo z zakresu ⟨**9, -9**⟩.

```
struct punkt
{
    int  nr, x, y;
};
```

4. Literatura

[1] Prata S.: Język C. Szkoła programowania. Wydanie VI. Helion, Gliwice, 2016.

[2] Kernighan B.W., Ritchie D.M.: Język ANSI C. Programowanie. Wydanie II. Helion, Gliwice, 2010.

[3] Prinz P., Crawford T.: Język C w pigułce. APN Promise, Warszawa, 2016.

[4] King K.N.: Język C. Nowoczesne programowanie. Wydanie II. Helion, Gliwice, 2011.

[5] Kochan S.G.: Język C. Kompendium wiedzy. Wydanie IV. Helion, Gliwice, 2015.

5. Pytania kontrolne

1. Omów sposób deklarowania struktur w języku C.
2. W jaki sposób można odwoływać się do pól struktury?
3. Opisz inicjalizację zmiennych strukturalnych.
4. Omów sposób deklarowania oraz zastosowanie pól bitowych i unii.

6. Wymagania BHP

Warunkiem przystąpienia do praktycznej realizacji ćwiczenia jest zapoznanie się z instrukcją BHP i instrukcją przeciw pożarową oraz przestrzeganie zasad w nich zawartych.

W trakcie zajęć laboratoryjnych należy przestrzegać następujących zasad.

- Sprawdzić, czy urządzenia dostępne na stanowisku laboratoryjnym są w stanie kompletnym, nie wskazującym na fizyczne uszkodzenie.
- Jeżeli istnieje taka możliwość, należy dostosować warunki stanowiska do własnych potrzeb, ze względu na ergonomię. Monitor komputera ustawić w sposób zapewniający stałą i wygodną obserwację dla wszystkich członków zespołu.
- Sprawdzić prawidłowość połączeń urządzeń.
- Załączenie komputera może nastąpić po wyrażeniu zgody przez prowadzącego.
- W trakcie pracy z komputerem zabronione jest spożywanie posiłków i picie napojów.

- W przypadku zakończenia pracy należy zakończyć sesję przez wydanie polecenia wylogowania. Zamknięcie systemu operacyjnego może się odbywać tylko na wyraźne polecenie prowadzącego.
- Zabronione jest dokonywanie jakichkolwiek przełączeń oraz wymiana elementów składowych stanowiska.
- Zabroniona jest zmiana konfiguracji komputera, w tym systemu operacyjnego i programów użytkowych, która nie wynika z programu zajęć i nie jest wykonywana w porozumieniu z prowadzącym zajęcia.
- W przypadku zaniku napięcia zasilającego należy niezwłocznie wyłączyć wszystkie urządzenia.
- Stwierdzone wszelkie braki w wyposażeniu stanowiska oraz nieprawidłowości w funkcjonowaniu sprzętu należy przekazywać prowadzącemu zajęcia.
- Zabrania się samodzielnego włączania, manipulowania i korzystania z urządzeń nie należących do danego ćwiczenia.
- W przypadku wystąpienia porażenia prądem elektrycznym należy niezwłocznie wyłączyć zasilanie stanowiska. Przed odłączeniem napięcia nie dotykać porażonego.