Politechnika Białostocka
Wydział Elektryczny
Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki

Instrukcja
do pracowni specjalistycznej z przedmiotu

Informatyka 2

Kod przedmiotu: **EZ1E3012**

(studia niestacjonarne)

JĘZYK C - WSKAŹNIKI, DYNAMICZNY PRZYDZIAŁ PAMIĘCI

Numer ćwiczenia

INF24Z

Autor:
dr inż. Jarosław Forenc

Białystok 2021

Spis treści

1. Opis stanowiska 3
1.1. Stosowana aparatura 3
1.2. Oprogramowanie 3
2. Wiadomości teoretyczne 3
2.1. Wskaźniki 3
2.2. Związek tablic ze wskaźnikami 7
2.3. Pamięć a zmienne w programie 9
2.4. Dynamiczny przydział pamięci w języku C 11
3. Przebieg ćwiczenia 15
4. Literatura 16
5. Pytania kontrolne 17
6. Wymagania BHP 17

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 10.

1.2. Oprogramowanie

Na komputerach zainstalowane jest środowisko programistyczne Code::Blocks.

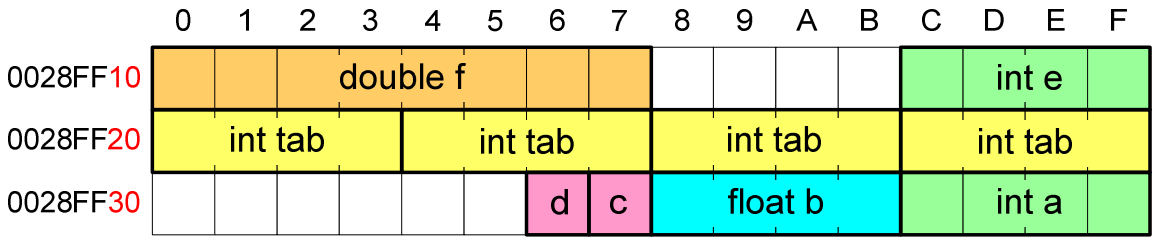
2. Wiadomości teoretyczne

2.1. Wskaźniki

Wskaźnik jest **zmienną** mogącą zawierać adres obszaru pamięci, a w szczególności adres innej zmiennej (obiektu). Załóżmy, że mamy następujące deklaracje zmiennych:

```
int a;
float b;
char c, d;
int tab[4];
int e;
double f;
```

Wszystkie zmienne przechowywane są w pamięci komputera. Zależnie od typu zmiennej zajmują określoną liczbę bajtów (Rys. 1).



Rys. 1. Zmienne w pamięci komputera

Każda zmienna oprócz nazwy ma także adres. Program nie posługuje się nazwami zmiennych tylko ich adresami. Na przykład zmienna **a** typu **int** zajmuje 4 bajty i znajduje się pod adresem **0028FF3C** (Rys. 1), tablica **tab** przechowująca cztery elementy typu **int**, znajduje się pod adresem **0028FF20** i zajmuje 4 × 4 = 16 bajtów. W języku C do wyświetlenia adresu zmiennej stosuje się specyfikator formatu **%p**.

```
printf("Adres zmiennej a:  %p\n", &a);
printf("Adres tablicy tab: %p\n", tab);
```

Adres określa miejsce w pamięci, natomiast nie informuje o rozmiarze wskazywanego obiektu. Deklarując wskaźnik (zmienną wskazującą) musimy podać typ obiektu, na jaki on wskazuje. Deklaracja zmiennej wskazującej wygląda tak samo, jak deklaracja każdej innej zmiennej, tylko że jej nazwa poprzedzona jest symbolem gwiazdki (*):

```
typ *nazwa_zmiennej;
```

lub

```
typ* nazwa_zmiennej;
```

lub

```
typ * nazwa_zmiennej;
```

lub

```
typ*nazwa_zmiennej;
```

Poniższa instrukcja zawiera deklarację zmiennej wskaźnikowej do typu **int**. Oznacza to, że zmienna **ptr** jest typu: **wskaźnik do zmiennej typu int**. Zmienna **ptr** może przechowywać adresy zmiennych **a** i **e** z wcześniejszego przykładu.

```
int *ptr;
```

Do przechowywania adresu zmiennej typu **double** należy zadeklarować zmienną typu: **wskaźnik do zmiennej typu double**.

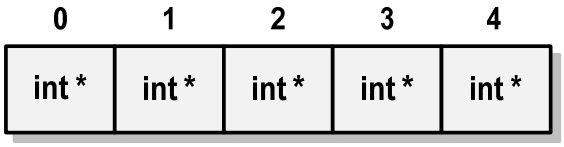
```
double *ptrd;
```

Można konstruować wskaźniki do danych dowolnego typu łącznie z typami **wskaźnik do wskaźnika do...** . W poniższym przykładzie **wsk** jest typu **wskaźnik do wskaźnika do typu char**.

```
char **wsk;
```

Można deklarować tablice wskaźników. Zmienna **tab_wsk** jest tablicą zawierającą 5 wskaźników do typu **int** (Rys. 2).

```
int *tab_wsk[5];
```



Rys. 2. Tablica zawierająca 5 wskaźników do typu **int**

Natomiast zmienna **wsk_tab** jest wskaźnikiem do 5-elementowej tablicy liczb **int**.

```
int (*wsk_tab)[5];
```

Deklarując wskaźniki lepiej jest pisać * przy zmiennej, a nie przy typie:

```
int *ptr1;    /* lepiej */
int* ptr2;    /* gorzej */
```

gdyż trudniej jest pomylić się przy deklaracji dwóch wskaźników:

```
int *p1, *p2;
int* p3, p4;
```

Zmienne **p1**, **p2** i **p3** są wskaźnikami do typu **int**, zaś zmienna **p4** jest „zwykłą” zmienną typu **int**.

Do przypisania wskaźnikowi wartości (czyli adresu) można zastosować jednoargumentowy operator pobierania adresu **&**:

```
int a = 10;    /* a - zmienna typu int */
int *ptr;      /* ptr - wskaźnik do typu int */
ptr = &a;
```

Poprzez adres zmiennej można „dostać się” do jej wartości używając tzw. **operatora wyłuskania** (**odwołania pośredniego**) - gwiazdki (*****):

```
*ptr = 20;
```

Jeśli zmienna **ptr** przechowuje adres zmiennej **a**, to powyższe przypisanie jest równoważne nadaniu zmiennej **a** wartości **20**:

```
a = 20;
```

Wskaźnik pusty to specjalna wartość, odróżnialna od wszystkich innych wartości wskaźnikowych, dla której gwarantuje się nierówność ze wskaźnikiem do dowolnego obiektu. Do zapisu wskaźnika pustego stosuje się wyrażenie całkowite o wartości zero (**0**):

```
int *ptr = 0;
```

Często zamiast wartości **0** stosowana jest makrodefinicja preprocesora **NULL**, która podczas kompilacji programu zamieniana jest na **0**:

```
int *ptr = NULL;
```

Przypisywanie wartości wskaźnikom.

```
#include <stdio.h>

int main(void)
{
   int x = 15;
   int *ptri = NULL;

   printf("x =    %d\n",x);
   printf("ptri = %p\n",ptri);

   ptri = &x;                    // przypisanie adresu
   printf("ptri = %p\n",ptri);

   *ptri = *ptri + 10;           // x = x + 10
   printf("x =    %d\n",x);
   printf("x =    %d\n",*ptri);

   return 0;
}
```

Przykładowy wynik uruchomienia programu:

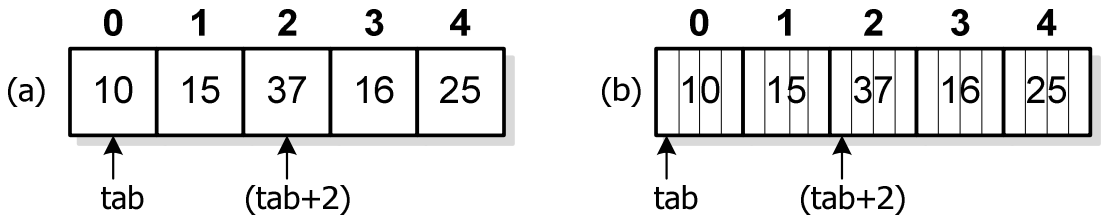
```
x =    15
ptri = 0000000000000000
ptri = 0000000000D5FB74
x =    25
x =    25
```

2.2. Związek tablic ze wskaźnikami

W języku C istnieje ścisły związek tablic ze wskaźnikami. Załóżmy, że dana jest następująca deklaracja tablicy i jej inicjalizacja:

```
int tab[5] = {10,15,37,16,25};
```

Nazwa tablicy jest adresem zerowego elementu tablicy (Rys. 3a). Dokładniej mówiąc - adresem pierwszego bajtu zajmowanego przez ten element (Rys. 3b).



Rys. 3. Odwołania do elementów tablicy

Jeśli nazwa tablicy jest adresem zerowego jej elementu, to stawiając przed nią symbol * możemy „dostać się” do wartości tego elementu. Zatem:

`*tab`	jest równoważne:	`tab[0]`

Podobnie:

`*(tab+1)`
`*(tab+2)`
`...`
	jest równoważne:	`tab[1]`
		`tab[2]`
		`...`

oraz ogólnie:

`*(tab+i)`	jest równoważne:	`tab[i]`

Przed dodaniem do wskaźnika liczby całkowitej jest ona mnożona przez liczbę bajtów zajmowanych przez wartość wskazywanego typu.

W zapisie ***(tab+i)** nawiasy są konieczne, gdyż operator * ma wyższy priorytet niż operator **+**. Poniższy przykład pokazuje różnice w zapisie z nawiasami i bez nawiasów.

```
int tab[5] = {10,15,37,16,25}, x;
x = *(tab+2);
printf("x = %d",x);              /*  x = 37 */
x = *tab+2;
printf("x = %d",x);              /*  x = 12 */
```

W pierwszym przypadku zmiennej **x** przypisywana jest wartość **37**, gdyż taką wartość ma element tablicy **tab** o indeksie **2** (zapis: **x = *(tab+2)** jest równoważny zapisowi: **x = tab[2]**). W drugim przypadku zmiennej **x** zostanie przypisana suma elementu tablicy **tab** o indeksie **0** i liczby **2** (zapis: **x = *tab+2** jest równoważny zapisowi: **x = tab[0] + 2**).

2.3. Pamięć a zmienne w programie

Ze względu na czas życia wyróżnia się w programie w języku C:

- **obiekty statyczne** - istnieją od chwili rozpoczęcia działania programu aż do jego zakończenia;
- obiekty **dynamiczne** - tworzone i usuwane z pamięci w trakcie wykonania programu - automatycznie (bez udziału programisty) lub kontrolowane przez programistę.

O typie obiektu (statyczny lub dynamiczny) decyduje **miejsce deklaracji** obiektu w kodzie programu oraz **klasa pamięci** obiektu. Ze względu na miejsce deklaracji obiektu w kodzie programu wyróżnia się:

- zmienne **globalne** (deklarowane poza funkcjami), które są statyczne;
- zmienne **lokalne** (deklarowane wewnątrz bloków funkcyjnych), których zaliczenie do obiektów statycznych lub dynamicznych zależy od klasy pamięci.

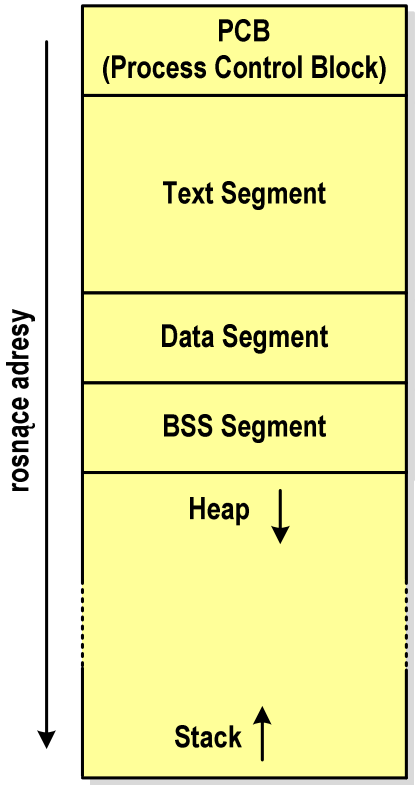
Klasę pamięci określają słowa kluczowe języka C:

- **auto** - zmienna automatyczna;
- **static** - zmienna statyczna;
- **register** - zmienna umieszczana w rejestrach procesora (automatyczna).

```
int x;              /* zmienna automatyczna */
static int y;       /* zmienna statyczna    */
register int z;     /* zmienna automatyczna */
```

Wszystkie zmienne lokalne zadeklarowane bez jawnego specyfikowania klasy pamięci są automatyczne (**auto**).

Kiedy uruchamiany jest program komputerowy to jest on ładowany do pamięci operacyjnej komputera. Struktura programu w pamięci przedstawiona jest na poniższym rysunku.



Rys. 4. Struktura programu w pamięci komputera

Blok kontrolny procesu (*PCB - Process Control Block*) jest obszarem pamięci operacyjnej zarezerwowanym przez system operacyjny do zarządzania procesem. **Segment kodu** (*Text Segment*) zawiera kod programu czyli instrukcje w postaci binarnej. **Segment danych** składa się z dwóch części. W pierwszej części (*Data Segment*) umieszczane są zmienne globalne i statyczne zainicjalizowane niezerowymi wartościami. W drugiej części (*BSS Segment - Block Started by Symbol*) znajdują się także zmienne globalne i statyczne, ale domyślnie zainicjalizowane zerowymi wartościami. **Stos** (*Stack*) przechowuje zmienne lokalne (automatyczne) oraz parametry i adresy powrotu z funkcji - ramki stosu (*Stack Frame*). **Sterta** (*Heap*) jest obszarem zmiennych dynamicznych. Stos programu jest ograniczony co do rozmiaru. Jeżeli program wymaga zadeklarowania bardzo dużej tablicy, to nie wykonuje się tego na stosie, ale na stercie. W takim przypadku należy zastosować dynamiczny przydział pamięci.

2.4. Dynamiczny przydział pamięci w języku C

Do dynamicznego przydziału pamięci w języku C stosowane są dwie funkcje: **calloc()** i **malloc()**. Przydział bloków pamięci następuje w obszarze sterty (stosu zmiennych dynamicznych). Przydzieloną dynamicznie pamięć należy zwolnić wywołując funkcję **free()**. Zastosowanie wymienionych funkcji wymaga dołączenia pliku nagłówkowego **stdlib.h**.

`calloc()`	Nagłówek: void ***calloc(size_t n, size_t size);**

- funkcja **calloc()** przydziela blok pamięci o rozmiarze **n*size** (mogący pomieścić tablicę **n**-elementów rozmiaru **size** każdy) i zwraca wskaźnik do tego bloku;
- jeśli nie można przydzielić pamięci, to funkcja zwraca wartość **NULL**;
- przydzielona pamięć jest inicjalizowana zerami (bitowo).

`malloc()`	Nagłówek: void ***malloc(size_t size);**

- funkcja **malloc()** przydziela blok pamięci o rozmiarze określonym parametrem **size** i zwraca wskaźnik do tego bloku;
- jeśli pamięci nie można przydzielić, to funkcja zwraca wartość **NULL**;
- przydzielona pamięć nie jest inicjalizowana.

Jeśli przydzielony blok pamięci nie jest już potrzebny, to należy zwolnić pamięć wywołując funkcję **free()**.

`free()`	Nagłówek: void ***free(**void ***ptr);**

- funkcja **free()** zwalnia blok pamięci wskazywany parametrem **ptr**;
- wartość **ptr** musi być wynikiem wywołania funkcji **calloc()** lub **malloc()**.

W poniższym programie przydzielana jest dynamicznie pamięć na jedną zmienną typu **float**.

Dynamiczny przydział pamięci na jedną zmienną.

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

int main(void)
{
   float *wsk;

   wsk = (float *) calloc(1,sizeof(float));
   if (wsk == NULL)
   {
      printf("Blad przydzialu pamieci\n");
      return 0;
   }

   printf("Podaj liczbe: ");
   scanf("%f",wsk);
   printf("Wartosc = %g\n",*wsk);

   free(wsk);
   return 0;
}
```

Przykładowy wynik uruchomienia programu:

```
Podaj liczbe: 25.18
Wartosc = 25.18
```

Do przydzielenia pamięci zastosowano funkcję **calloc()**. Funkcja ta zwraca wskaźnik do typu **void**, dlatego wartość wskaźnika należy rzutować na właściwy typ (**float** *). Jeśli nie było możliwe przydzielenie pamięci (**wsk == NULL**), to wyświetlany jest odpowiedni komunikat i program kończy pracę. W przeciwnym przypadku wczytywana jest wartość liczby typu **float**, a następnie jest ona wyświetlana na ekranie. Przed zmienną **wsk** nie ma znaku **&**, gdyż zmienna ta jest już wskaźnikiem (czego wymaga funkcja **scanf()**). Na koniec funkcja **free()** zwalnia pamięć przydzieloną na zmienną.

Kolejny przykład przedstawia dynamiczny przydział pamięci na tablicę jednowymiarową (wektor) liczb całkowitych typu **int**.

Dynamiczny przydział pamięci na tablicę jednowymiarową.

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

int main(void)
{
   int *tab, n = 10;

   tab = (int *) malloc(n*sizeof(int));
   if (tab == NULL)
   {
      printf("Blad przydzialu pamieci\n");
      return 0;
   }

   for (int i=0; i<n; i++)
   {
      tab[i] = i*i;
      printf("tab[%d] = %d\n",i,tab[i]);
   }

   free(tab);

   return 0;
}
```

Wynik uruchomienia programu:

```
tab[0] = 0
tab[1] = 1
tab[2] = 4
tab[3] = 9
tab[4] = 16
tab[5] = 25
tab[6] = 36
tab[7] = 49
tab[8] = 64
tab[9] = 81
```

Do przydziału pamięci zastosowano funkcję **malloc()**. Funkcja ta zwraca wskaźnik do typu **void**, dlatego wartość wskaźnika należy rzutować na właściwy typ (**int ***). Po przydzieleniu pamięci następuje zapisanie do tablicy kwadratów kolejnych liczb i ich wyświetlenie na ekranie. Sposób odwoływania się do elementów tablicy jest identyczny, jak w przypadku zwykłej tablicy (**tab[i]**). Na koniec funkcja **free()** zwalnia pamięć przydzieloną na tablicę. Przy odwoływaniu się do elementów tablicy można zastosować także operator odwołania pośredniego * (gwiazdka).

```
for (int i=0; i<n; i++)
{
   *(tab+i) = i*i;
   printf("tab[%d] = %d\n",i,*(tab+i));
}
```

W poniższym przykładzie pokazano sposób dynamicznego przydziału pamięci na zmienną strukturalną i sposób odwoływania się do pól struktury.

Dynamiczny przydział pamięci na zmienną strukturalną.

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

struct punkt
{
   float x, y;
};

int main(void)
{
   struct punkt *wskp;
   float odl;

   wskp = (struct punkt*) malloc(sizeof(struct punkt));
   if (wskp == NULL)
   {
      printf("Blad przydzialu pamieci\n");
      return 0;
   }
```

```
    wskp->x = 10; wskp->y = -10;
    odl = sqrt(wskp->x*wskp->x + wskp->y*wskp->y);
    printf("Odleglosc od punktu(0,0): %g\n",odl);

    free(wskp);

    return 0;
}
```

Wynik uruchomienia programu:

```
Odleglosc od punktu(0,0): 14.1421
```

3. Przebieg ćwiczenia

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

1. Zadeklaruj w programie zmienne typów: **float**, **char**, **double**, **int**, tablicę 10 elementów typu **char**, zmienną typu **short**. Wyświetl adresy wszystkich zadeklarowanych zmiennych (zastosuj specyfikator formatu **%p**). Na podstawie adresów określ:
   a) czy zmienne znajdują się w pamięci komputera w takiej samej kolejności, jak w deklaracji?
   b) czy zmienne znajdują się w pamięci komputera bezpośrednio po sobie?

2. Napisz program, w którym użytkownik wprowadza z klawiatury liczbę elementów wektora. Przydziel dynamicznie pamięć na wektor liczb całkowitych. Wykonaj następujące operacje:
   a) zapisz do wektora wygenerowane pseudolosowo liczby całkowite z przedziału ⟨**0, 99**⟩;
   b) wyświetl na ekranie elementy wektora;
   c) oblicz i wyświetl **sumę** oraz **średnią arytmetyczną** elementów wektora;
   d) znajdź i wyświetl wartość elementu o **największej** wartości.

3. Napisz program, w którym użytkownik wprowadza z klawiatury stopień wielomianu **w(x)**. Przydziel dynamicznie pamięć na wektor przechowujący współczynniki tego wielomianu. Wczytaj wartości współczynników z klawiatury i zapisz je w wektorze. Oblicz wartość wielomianu dla argumentu **x** wczytanego z klawiatury. Zastosuj schemat Hornera. Zwolnij pamięć.

4. Zdefiniuj strukturę opisującą wybrane **urządzenie elektryczne**. Struktura powinna składać się z min. trzech pól różnych typów. Przydziel dynamicznie pamięć na zmienną strukturalną, zapisz wartości do pól struktury (przypisanie lub wczytanie z klawiatury), wyświetl zawartość pól struktury, zwolnij pamięć.

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] Deitel P.J., Deitel H.: Język C. Solidna wiedza w praktyce. Wydanie VIII. Helion, Gliwice, 2020.

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

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

[6] Reese R.: Wskaźniki w języku C. Przewodnik. Helion, Gliwice, 2014.

[7] Reek K.A.: Język C. Wskaźniki. Vademecum profesjonalisty. Helion, Gliwice, 2003.

[8] http://www.cplusplus.com/reference/clibrary - C library - C++ Reference

[9] https://cpp0x.pl/dokumentacja/standard-C/1 - Standard C

[10] https://www.codeblocks.org/ - Code::Blocks

5. Pytania kontrolne

1. Wyjaśnij pojęcie wskaźnika, podaj jak deklaruje się wskaźniki.
2. Jaki jest związek tablic ze wskaźnikami w języku C?
3. Scharakteryzuj obiekty statyczne i dynamiczne występujące w kodzie programu.
4. Opisz funkcje do dynamicznego przydzielania i zwalniania pamięci w języku C.

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.