Informatyka 1 (EZ1F1002)

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr II, studia niestacjonarne I stopnia

Rok akademicki 2022/2023

Wykład nr 8 (18.12.2022)

dr inż. Jarosław Forenc

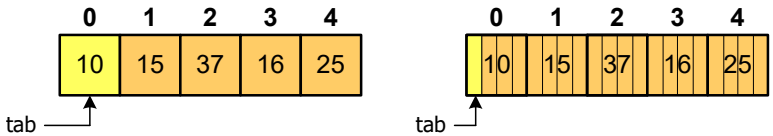
Plan wykładu nr 8

- Wskaźniki
  - związek z tablicami
  - dynamiczny przydział pamięci
- Funkcje w języku C
  - ogólna strukturafunkcji
  - argumenty i parametry funkcji
  - domyślne wartości parametrów funkcji
  - prototypy funkcji

Wskaźniki a tablice

- Nazwa tablicy jest jej adresem (dokładniej - adresem elementu o indeksie 0)

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

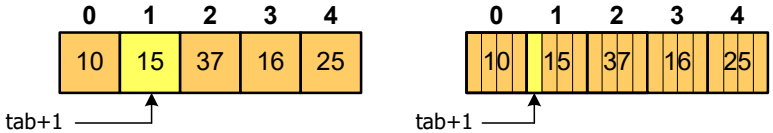


- Zastosowanie operatora * przed nazwą tablicy pozwala „dostać się" do zawartości elementu o indeksie 0

*tab jest równoważne tab[0]

Wskaźniki a tablice

- Dodanie 1 do adresu tablicy przenosi nas do elementu tablicy o indeksie 1



zatem: *(tab+1) jest równoważne tab[1]

ogólnie: *(tab+i) jest równoważne tab[i]

- W zapisie *(tab+i) nawiasy są konieczne, gdyż operator * ma bardzo wysoki priorytet

Wskaźniki a tablice

- Brak nawiasów powoduje błędne odwołania do elementów tablicy

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

x = *(tab+2);
printf("x = %d",x);          /*  x = 37 */

x = *tab+2;
printf("x = %d",x);          /*  x = 12 */
```

x = *(tab+2); jest równoważne x = tab[2];

x = *tab+2; jest równoważne x = tab[0]+2;

Dynamiczny przydział pamięci w języku C

- Kiedy stosuje się dynamiczny przydział pamięci?
  - gdy rozmiar tablicy będzie znany dopiero podczas wykonania programu a nie podczas jego kompilacji
  - gdy rozmiar tablicy jest bardzo duży
- Do dynamicznego przydziału pamięci stosowane są funkcje:
  - calloc()
  - malloc()
- Przydział pamięci następuje w obszarze sterty (stosu zmiennych dynamicznych)
- Przydzieloną pamięć należy zwolnić wywołując funkcję:
  - free()

Dynamiczny przydział pamięci w języku C

CALLOC stdlib.h

```
void *calloc(size_t num, size_t size);
```

- Przydziela blok pamięci o rozmiarze num*size (mogący pomieścić tablicę num-elementów, każdy rozmiaru size)
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość NULL
- Przydzielona pamięć jest inicjowana zerami (bitowo)
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;
tab = (int *) calloc(10,sizeof(int));
```


Dynamiczny przydział pamięci w języku C

MALLOC stdlib.h

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

- Przydziela blok pamięci o rozmiarze określonym parametrem size
- Zwraca wskaźnik do przydzielonego bloku pamięci
- Jeśli pamięci nie można przydzielić, to zwraca wartość NULL
- Przydzielona pamięć nie jest inicjowana
- Zwracaną wartość wskaźnika należy rzutować na właściwy typ

```
int *tab;
tab = (int *) malloc(10*sizeof(int));
```

Dynamiczny przydział pamięci w języku C

FREE stdlib.h

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

- Zwalnia blok pamięci wskazywany parametrem ptr
- Wartość ptr musi być wynikiem wywołania funkcji calloc() lub malloc()

```
int *tab;
tab = (int *) calloc(10,sizeof(int));
/* ... */
free(tab);
```


Przykład: 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;
   }

   *wsk = 123.45f;
   printf("wartosc = %g\n",*wsk);

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

```
wartosc = 123.45
```


Przykład: przydział pamięci na strukturę

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

struct punkt
{
   int x, y;
};

int main(void)
{
   struct punkt p, *wsk_p;

   wsk_p = (struct punkt*) malloc(sizeof(struct punkt));

   p.x = 10; p.y = 20;
   wsk_p->x = 30; wsk_p->y = 40;
   printf("%d,%d - %d,%d\n",p.x,p.y,wsk_p->x,wsk_p->y);

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

```
10,20 - 30,40
```


Przykład: przydział pamięci na wektor

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

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

   tab = (int *) calloc(n,sizeof(int));

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

   free(tab);

   return 0;
}
```

```
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
```

Program w języku C

- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

```
#include <stdio.h>    /* przekątna kwadratu */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Bok = %g, przekatna = %g\n",a,d);

    return 0;
}
```

```
Bok = 10, przekatna = 14.1421
```


Program w języku C

- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

```
#include <stdio.h>    /* przekątna kwadratu */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Bok = %g, przekatna = %g\n",a,d);

    return 0;
}
```

definicja funkcji

Program w języku C

- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

```
#include <stdio.h>    /* przekątna kwadratu */
#include <math.h>

int main(void)
{
    float a = 10.0f, d;

    d = a * sqrt(2.0f);
    printf("Bok = %g, przekatna = %g\n",a,d);

    return 0;
}
```

wywołania funkcji

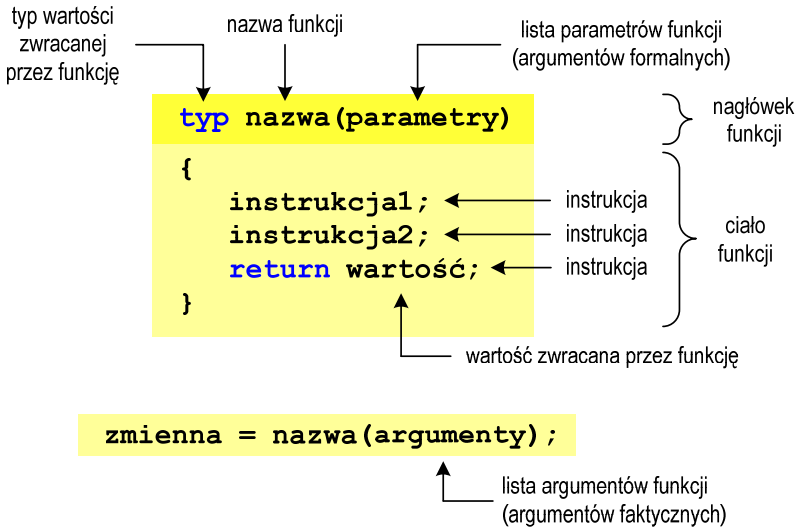
Funkcje w języku C

```
#include <stdio.h>    /* przekątna kwadratu */
#include <math.h>

float przekatna(float bok)
{
    float wynik;
    wynik = bok * sqrt(2.0f);
    return wynik;
}

int main(void)
{
    float a = 10.0f, d;
    d = przekatna(a);
    printf("Bok = %g, przekatna = %g\n",a,d);
    return 0;
}
```

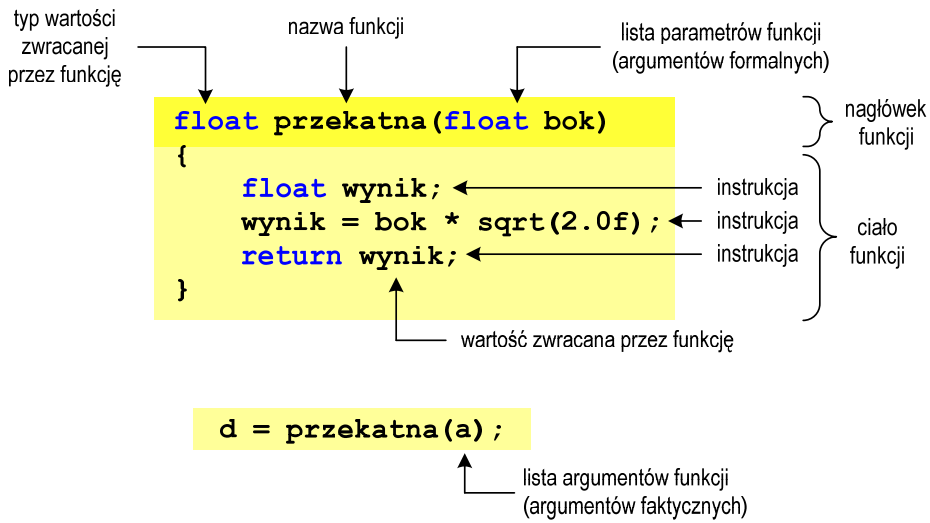
definicja funkcji

definicja funkcji

Ogólna struktura funkcji w języku C



Ogólna struktura funkcji w języku C



Argumenty funkcji

- Argumentami funkcji mogą być stałe liczbowe, zmienne, wyrażenia arytmetyczne, wywołania innych funkcji

```
d = przekatna(a);
d = przekatna(10);
d = przekatna(2*a+5);
d = przekatna(sqrt(a)+15);
```

- Wywołanie funkcji może być argumentem innej funkcji

```
printf("Bok = %g, przekatna = %g\n",
                              a, przekatna(a));
```


Parametry funkcji

- Parametry funkcji traktowane są tak samo jak zmienne zadeklarowane w tej funkcji i zainicjalizowane wartościami argumentów wywołania

```
float przekatna(float bok)
{
    float wynik;
    wynik = bok * sqrt(2.0f);
    return wynik;
}
```

- Funkcję przekatna() można zapisać w prostszej postaci:

```
float przekatna(float bok)
{
    return bok * sqrt(2.0f);
}
```

Parametry funkcji

- Jeśli funkcja ma kilka parametrów, to dla każdego z nich podaje się:
  - typ parametru
  - nazwę parametru
- Parametry oddzielane są od siebie przecinkami

```
/* przekątna prostokąta */

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}
```


Parametry funkcji

- W różnych funkcjach zmienne mogą mieć takie same nazwy

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}

int main(void)
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```


Domyślne wartości parametrów funkcji

- W definicji funkcji można jej parametrom nadać domyślne wartości

```
float przekatna(float a = 10, float b = 5.5f)
{
    return sqrt(a*a+b*b);
}
```

- W takim przypadku funkcję można wywołać z dwoma, jednym lub bez żadnych argumentów

```
d = przekatna(a,b);
```

```
d = przekatna(a);
```

```
d = przekatna();
```

- Brakujące argumenty zostaną zastąpione wartościami domyślnymi

Domyślne wartości parametrów funkcji

- Nie wszystkie parametry muszą mieć podane domyślne wartości
- Wartości muszą być podawane od prawej strony listy parametrów

```
float przekatna(float a, float b = 5.5f)
{
    return sqrt(a*a+b*b);
}
```

- Powyższa funkcja może być wywołana z jednym lub dwoma argumentami

```
d = przekatna(a,b);
```

```
d = przekatna(a);
```

- Domyślne wartości parametrów mogą być podane w deklaracji lub w definicji funkcji

Wartość zwracana przez funkcję

- Słowo kluczowe return może wystąpić w funkcji wiele razy

```
float ocena(int pkt)
{
    if (pkt>90)            return 5.0f;
    if (pkt>80 && pkt<91)  return 4.5f;
    if (pkt>70 && pkt<81)  return 4.0f;
    if (pkt>60 && pkt<71)  return 3.5f;
    if (pkt>50 && pkt<61)  return 3.0f;
    if (pkt<51)            return 2.0f;
}
```

91-100 pkt. → 5,0	81-90 pkt. → 4,5
71-80 pkt. → 4,0	61-70 pkt. → 3,5
51-60 pkt. → 3,0	0-50 pkt. → 2,0

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <stdio.h>    /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b)          definicja funkcji
{
    return sqrt(a*a+b*b);
}

int main(void)                             definicja funkcji
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```


Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <stdio.h>    /* przekątna prostokąta */
#include <math.h>

int main(void)                             definicja funkcji
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}

float przekatna(float a, float b)          definicja funkcji
{
    return sqrt(a*a+b*b);
}
```


Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <stdio.h>    /* przekątna prostokąta */
#include <math.h>

int main(void)                             definicja funkcji
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prosto...ta = %g\n",d);
    return 0;
}

float przekatna(float a, float b)
{
    return sqrt(a*a+b*b);
}
```

error C3861: 'przekatna': identifier not found

Prototyp funkcji

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b);                    prototyp funkcji

int main(void)                                        definicja funkcji
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}

float przekatna(float a, float b)                     definicja funkcji
{
    return sqrt(a*a+b*b);
}
```


Prototyp funkcji

- Prototyp funkcji jest to jej nagłówek zakończony średnikiem

```
float przekatna(float a, float b);
```

- Inne określenia prototypu funkcji:
  - deklaracja funkcji
  - zapowiedź funkcji
- Dzięki prototypowi kompilator sprawdza w wywołaniu funkcji:
  - nazwę funkcji
  - liczbę i typ argumentów
  - typ zwracanej wartości

```
d = przekatna(a,b);
```

- Nazwy parametrów nie mają znaczenia i mogą być pominięte:

```
float przekatna(float, float);
```


Prototyp funkcji

- W przypadku umieszczenia prototypu funkcji i pominięcia jej definicji błąd wystąpi nie na etapie kompilacji, ale łączenia (linkowania)

```
#include <stdio.h>     /* przekątna prostokąta */
#include <math.h>

float przekatna(float a, float b);                    prototyp funkcji

int main(void)                                        definicja funkcji
{
    float a = 10.0f, b = 5.5f, d;
    d = przekatna(a,b);
    printf("Przekatna prostokata = %g\n",d);
    return 0;
}
```


Prototyp funkcji

- W przypadku umieszczenia prototypu funkcji i pominięcia jej definicji błąd wystąpi nie na etapie kompilacji, ale łączenia (linkowania)

```
1>Compiling...
1>test.cpp
1>Compiling manifest to resources...
1>Microsoft (R) Windows (R) Resource Compiler Version 6.0.5724.0
1>Copyright (C) Microsoft Corporation.  All rights reserved.
1>Linking...
1>test.obj : error LNK2019: unresolved external symbol "float __cdecl
przekatna(float,float)" (?przekatna@@YAMMM@Z) referenced in function _main
1>D:\test\Debug\test.exe : fatal error LNK1120: 1 unresolved externals
```

Koniec wykładu nr 8

Dziękuję za uwagę!