Informatyka 2

Politechnika Białostocka - Wydział Elektryczny

Elektrotechnika, semestr III, studia stacjonarne I stopnia

Rok akademicki 2018/2019

Wykład nr 5 (30.10.2018)

dr inż. Jarosław Forenc

Plan wykładu nr 5

- Funkcje w języku C
  - wskaźniki do funkcji
  - wywołanie funkcji przez wskaźnik
- Prototypy funkcji, typy funkcji
- Przekazywanie argumentów do funkcji
  - przez wartość i przez wskaźnik
  - przekazywanie wektorów, macierzy i struktur
  - const przed parametrem funkcji
- Pamięć a zmienne w programie
  - zmienna automatyczne (auto)
  - zmienne rejestrowe (register)

Funkcje w języku C

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

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

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


Wskaźniki do funkcji

- Definicja funkcji

```
typ nazwa_funkcji(parametry)
{
}
```

- Można deklarować wskaźniki do funkcji

```
typ (*nazwa_wskaźnika)(parametry);
```

- Przykłady deklaracji funkcji i odpowiadającym im wskaźników

```
void foo();
int foo(double x);
void foo(char *x);
int *foo(int x,int y);
float *foo(void);
```

```
void (*fptr)();
int (*fptr)(double);
void (*fptr)(char *);
int *(*fptr)(int,int);
float *(*fptr)(void);
```

Wywołanie funkcji przez wskaźnik

```
#include <stdio.h>

int suma(int x, int y)
{
    return x + y;
}

int main(void)
{
    int (*fptr)(int,int);      // deklaracja wskaźnika do funkcji
    int w;

    fptr = suma;               // przypisanie wskaźnikowi adresu funkcji
    w = fptr(5,10);            // wywołanie funkcji przez wskaźnik
    printf("w = %d\n",w);

    return 0;
}
```

```
w = 15
```


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[illegible]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
```

Typy funkcji (1)

- Dotychczas prezentowane funkcje miały argumenty i zwracały wartości
- Struktura i wywołanie takiej funkcji ma następującą postać

```
typ nazwa(parametry)
{
   instrukcje;
   return wartość;
}
```

```
typ zm;
zm = nazwa(argumenty);
```

- Można zdefiniować także funkcje, które nie mają argumentów i/lub nie zwracają żadnej wartości

Typy funkcji (2)

- Funkcja bez argumentów i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to void
  - zamiast parametrów, podaje się słowo void lub nie wpisuje się nic
  - jeśli występuje return, to nie może po nim znajdować się żadna wartość
  - jeśli return nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(void)
{
   instrukcje;
   return;
}
```

```
void nazwa()
{
   instrukcje;
   return;
}
```


Typy funkcji (2)

- Funkcja bez argumentów i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to void
  - zamiast parametrów, podaje się słowo void lub nie wpisuje się nic
  - jeśli występuje return, to nie może po nim znajdować się żadna wartość
  - jeśli return nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(void)
{
   instrukcje;
}
```

```
void nazwa()
{
   instrukcje;
}
```

- Wywołanie funkcji:

```
nazwa();
```


Typy funkcji (2) - przykład

```
#include <stdio.h>

void drukuj_linie(void)
{
    printf("--------------------------\n");
}

int main(void)
{
    drukuj_linie();
    printf("Funkcje nie sa trudne!\n");
    drukuj_linie();

    return 0;
}
```

```
--------------------------
Funkcje nie sa trudne!
--------------------------
```

Typy funkcji (3)

- Funkcja z argumentami i nie zwracająca wartości:
  - w nagłówku funkcji, typ zwracanej wartości to void
  - jeśli występuje return, to nie może po nim znajdować się żadna wartość
  - jeśli return nie występuje, to funkcja kończy się po wykonaniu wszystkich instrukcji
- Struktura funkcji:

```
void nazwa(parametry)
{
   instrukcje;
   return;
}
```

```
void nazwa(parametry)
{
   instrukcje;
}
```

- Wywołanie funkcji: `nazwa(argumenty);`

Typy funkcji (3) - przykład

```
#include <stdio.h>

void drukuj_dane(char *imie, char *nazwisko, int wiek)
{
    printf("Imie:           %s\n",imie);
    printf("Nazwisko:       %s\n",nazwisko);
    printf("Wiek:           %d\n",wiek);
    printf("Rok urodzenia:  %d\n\n",2018-wiek);
}

int main(void)
{
    drukuj_dane("Jan","Kowalski",23);
    drukuj_dane("Barbara","Nowak",28);

    return 0;
}
```


Typy funkcji (3) - przykład

```
#include <stdio.h>

void drukuj_dane(char *imie,
{
    printf("Imie:
    printf("Nazwisko:
    printf("Wiek:
    printf("Rok urodzenia:
}

int main(void)
{
    drukuj_dane("Jan","Kowalski",23);
    drukuj_dane("Barbara","Nowak",28);

    return 0;
}
```

```
Imie:           Jan
Nazwisko:       Kowalski
Wiek:           23
Rok urodzenia:  1995

Imie:           Barbara
Nazwisko:       Nowak
Wiek:           28
Rok urodzenia:  1990
```


Typy funkcji (4)

- Funkcja bez argumentów i zwracająca wartość:
  - zamiast parametrów, podaje się słowo void lub nie wpisuje się nic
  - typ zwracanej wartości musi być zgodny z typem w nagłówku funkcji
- Struktura funkcji:

```
typ nazwa(void)
{
   instrukcje;
   return wartość;
}
```

```
typ nazwa()
{
   instrukcje;
   return wartość;
}
```

- Wywołanie funkcji:

```
typ zm;
zm = nazwa();
```

Typy funkcji (4) - przykład

```
W roku jest: 31536000 sekund
```

```c
#include <stdio.h>

int liczba_sekund_rok(void)
{
    return (365 * 24 * 60 * 60);
}

int main(void)
{
    int wynik;

    wynik = liczba_sekund_rok();
    printf("W roku jest: %d sekund\n",wynik);

    return 0;
}
```


Przekazywanie argumentów do funkcji

- Przekazywanie argumentów przez wartość:
  - po wywołaniu funkcji tworzone są lokalne kopie zmiennych skojarzonych z jej argumentami
  - w funkcji widoczne są one pod postacią parametrów funkcji
  - parametry te mogą być traktowane jak lokalne zmienne, którym przypisano początkową wartość
- Przekazywanie argumentów przez wskaźnik:
  - do funkcji przekazywane są adresy zmiennych będących jej argumentami
  - wszystkie operacje wykonywane w funkcji na takich argumentach będą odnosiły się do zmiennych z funkcji wywołującej

Przekazywanie argumentów przez wartość

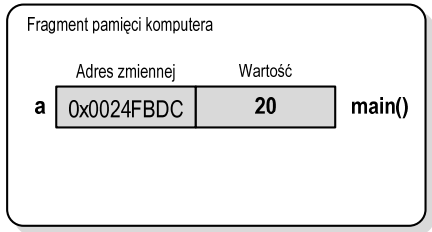
```c
#include <stdio.h>

void fun(int a)
{
   a = 10;
   printf("fun: a = %d\n",a
}

int main(void)
{
  int a = 20;

  fun(a);
  printf("main: a = %d\n",a);

  return 0;
}
```



Przekazywanie argumentów przez wartość

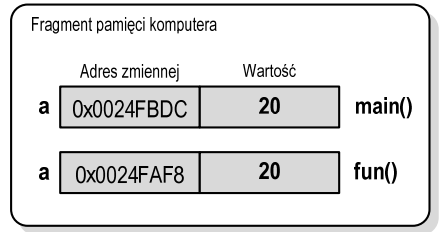
```c
#include <stdio.h>

void fun(int a)
{
   a = 10;
   printf("fun: a = %d\n",a
}

int main(void)
{
  int a = 20;

  fun(a);
  printf("main: a = %d\n",a);

  return 0;
}
```



Przekazywanie argumentów przez wartość

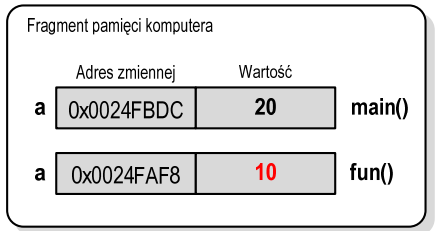
```
#include <stdio.h>

void fun(int a)
{
   a = 10;
   printf("fun: a = %d\n",a
}

int main(void)
{
  int a = 20;

  fun(a);
  printf("main: a = %d\n",a);

  return 0;
}
```




```
fun:  a = 10
```


Przekazywanie argumentów przez wartość

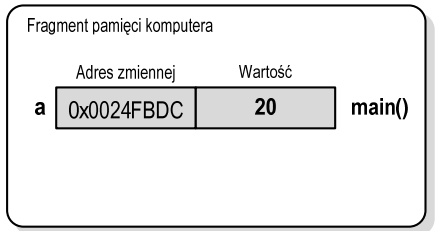
```
#include <stdio.h>

void fun(int a)
{
   a = 10;
   printf("fun: a = %d\n",a
}

int main(void)
{
  int a = 20;

  fun(a);
  printf("main: a = %d\n",a);

  return 0;
}
```




```
fun:  a = 10
main: a = 20
```


Przekazywanie argumentów przez wskaźnik

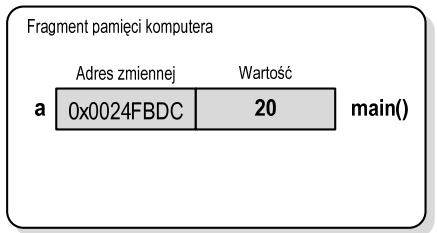
```
#include <stdio.h>

void fun(int *a)
{
   *a = 10;
   printf("fun: a = %d\n",*
}

int main(void)
{
  int a = 20;

  fun(&a);
  printf("main: a = %d\n",a);

  return 0;
}
```



Przekazywanie argumentów przez wskaźnik

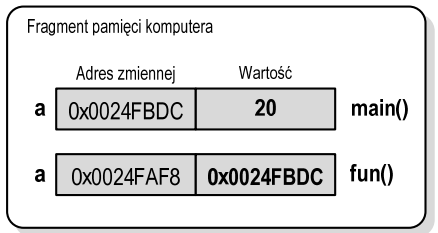
```
#include <stdio.h>

void fun(int *a)
{
   *a = 10;
   printf("fun: a = %d\n",*
}

int main(void)
{
  int a = 20;

  fun(&a);
  printf("main: a = %d\n",a);

  return 0;
}
```



Przekazywanie argumentów przez wskaźnik

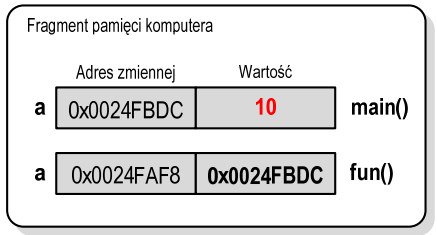
```
#include <stdio.h>

void fun(int *a)
{
   *a = 10;
   printf("fun: a = %d\n",*
}

int main(void)
{
  int a = 20;

  fun(&a);
  printf("main: a = %d\n",a);

  return 0;
}
```



fun: a = 10

Przekazywanie argumentów przez wskaźnik

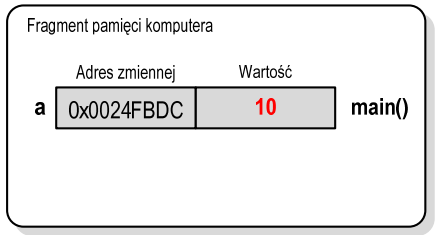
```
#include <stdio.h>

void fun(int *a)
{
   *a = 10;
   printf("fun: a = %d\n",*
}

int main(void)
{
  int a = 20;

  fun(&a);
  printf("main: a = %d\n",a);

  return 0;
}
```



fun: a = 10
main: a = 10

Parametry funkcji - wektory

- Wektory przekazywane są do funkcji przez wskaźnik
- Nie jest tworzona kopia tablicy, a wszystkie operacje na jej elementach odnoszą się do tablicy z funkcji wywołującej
- W nagłówku funkcji podaje się typ elementów tablicy, jej nazwę oraz nawiasy kwadratowe z liczbą elementów tablicy lub same nawiasy kwadratowe

```
void fun(int tab[5])
{
   ...
}
```

```
void fun(int tab[])
{
   ...
}
```

- W wywołaniu funkcji podaje się tylko jej nazwę (bez nawiasów kwadratowych)

```
fun(tab);
```


Parametry funkcji - wektory (przykład)

```
#include <stdio.h>

void drukuj(int tab[])
{
   for (int i=0; i<5; i++)
      printf("%3d",tab[i]);
   printf("\n");
}

void zeruj(int tab[5])
{
   for (int i=0; i<5; i++)
      tab[i] = 0;
}
```

```
float srednia(int tab[])
{
   float sr = 0;
   int suma = 0;

   for (int i=0; i<5; i++)
      suma = suma + tab[i];

   sr = (float)suma / 5;

   return sr;
}
```

Parametry funkcji - wektory (przykład)

```
int main(void)
{
    int tab[5] = {1,2,3,4,5};
    float sred;

    drukuj(tab);

    sred = srednia(tab);
    printf("Srednia elementow: %g\n", sred);
    printf("Srednia elementow: %g\n", srednia(tab));

    zeruj(tab);
    drukuj(tab);

    return 0;
}
```

```
 1  2  3  4  5
srednia elementow: 3
srednia elementow: 3
 0  0  0  0  0
```


Parametry funkcji - const

- Jeśli funkcja nie powinna zmieniać wartości przekazywanych do niej zmiennych, to w nagłówku, przed odpowiednim parametrem, dodaje się identyfikator const

```
void drukuj(const int tab[])
{
   for (int i=0; i<5; i++)
   {
      printf("%3d",tab[i]);
      tab[i] = 0;
   }
   printf("\n");
}
```

- Podczas kompilacji takiej funkcji wystąpi błąd

error C3892: 'tab' : you cannot assign to a variable that is const

Parametry funkcji - const

- Przykładowe prototypy funkcji z pliku nagłówkowego string.h

```
char* strcpy(char *dest, const char *source);
```

```
size_t strlen(const char *str);
```

```
char* strupr(char *str);
```


Parametry funkcji - macierze

- Macierze przekazywane są do funkcji przez wskaźnik
- W nagłówku funkcji podaje się typ elementów tablicy, jej nazwę oraz w nawiasach kwadratowych liczbę wierszy i kolumn lub tylko liczbę kolumn

```
void fun(int tab[2][3])
{
   ...
}
```

```
void fun(int tab[][3])
{
   ...
}
```

- W wywołaniu funkcji podaje się tylko jej nazwę (bez nawiasów kwadratowych)

```
fun(tab);
```

Parametry funkcji - macierze (przykład)

```
#include <stdio.h>

void zero(int tab[][3])
{
   for (int i=0; i<2; i++)
      for (int j=0; j<3; j++)
         tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
   for (int i=0; i<2; i++)
   {
      for (int j=0; j<3; j++)
         printf("%3d",tab[i][j]);
      printf("\n");
   }
}
```

```
int main(void)
{
   int tab[2][3] =
       {1,2,3,4,5,6};

   drukuj(tab);
   zero(tab);
   printf("\n");
   drukuj(tab);

   return 0;
}
```


Parametry funkcji - macierze (przykład)

```
#include <stdio.h>

void zero(int tab[][3])
{
   for (int i=0; i<2; i++)
      for (int j=0; j<3; j++)
         tab[i][j] = 0;
}

void drukuj(int tab[2][3])
{
   for (int i=0; i<2; i++)
   {
      for (int j=0; j<3; j++)
         printf("%3d",tab[i][j]);
      printf("\n");
   }
}
```

```
int main
{
   int t
       {

   druku
   zero(
   printf("\n");
   drukuj(tab);

   return 0;
}
```

```
  1  2  3
  4  5  6

  0  0  0
  0  0  0
```


Parametry funkcji - struktury

- Struktury przekazywane są do funkcji przez wartość (nawet jeśli daną składową jest tablica)

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

struct pkt
{
    float x, y;
};

float odl(struct pkt pkt1, struct pkt pkt2)
{
     return sqrt(pow(pkt2.x-pkt1.x,2)+
            pow(pkt2.y-pkt1.y,2));
}
```


Parametry funkcji - struktury (przykład)

```
int main(void)
{
    struct pkt p1 = {2,3};
    struct pkt p2 = {-2,1};
    float wynik;

    wynik = odl(p1,p2);

    printf("Punkt nr 1: (%g,%g)\n",p1.x,p1.y);
    printf("Punkt nr 2: (%g,%g)\n",p2.x,p2.y);
    printf("Odleglosc = %g\n",wynik);

    return 0;
}
```

```
Punkt nr 1: (2,3)
Punkt nr 2: (-2,1)
Odleglosc = 4.47214
```

Pamięć a zmienne w programie

- Ze względu na czas życia wyróżnia się w programie:
  - 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)
    - kontrolowane przez programistę
- O typie obiektu (statyczny lub dynamiczny) decyduje klasa pamięci obiektu (ang. storage class)
  - auto - zmienne automatyczne
  - register - zmienne umieszczane w rejestrach procesora
  - extern - zmienne zewnętrzne
  - static - zmienne statyczne

Zmienne automatyczne - auto

- Miejsce deklaracji: najczęściej początek bloku funkcyjnego ograniczonego nawiasami klamrowymi { i }
- Pamięć przydzielana automatycznie przy wejściu do bloku i zwalniana po wyjściu z niego
- Zakres widzialności: ograniczony do bloku, w którym zmienne zostały zadeklarowane (zmienne lokalne)
- Dostęp do zmiennych z innych bloków możliwy przez wskaźnik
- Jeśli zmienne są inicjalizowane, to odbywa się ona przy każdym wejściu do bloku, w którym zostały zadeklarowane
- Nie ma potrzeby jawnego używania auto, gdyż domyślnie zmienne wewnątrz bloków funkcyjnych są lokalne

```
auto int x;
```


Zmienne rejestrowe - register

- Zazwyczaj o miejscu umieszczenia zmiennej automatycznej decyduje kompilator:
  - pamięć operacyjna - wolniejszy dostęp
  - rejestry procesora - szybszy dostęp
- Programista może zasugerować kompilatorowi umieszczenie określonej zmiennej automatycznej w rejestrach procesora
- Najczęściej dotyczy to zmiennych:
  - często używanych
  - takich, dla których czas dostępu jest bardzo ważny

```
register int x;
```


Koniec wykładu nr 5

Dziękuję za uwagę!