Programowanie obiektowe (TZ1E2010)

Politechnika Białostocka - Wydział Elektryczny

Elektronika i telekomunikacja, semestr II
studia niestacjonarne I stopnia

Rok akademicki 2020/2021

Pracownia nr 2 (12.03.2021)

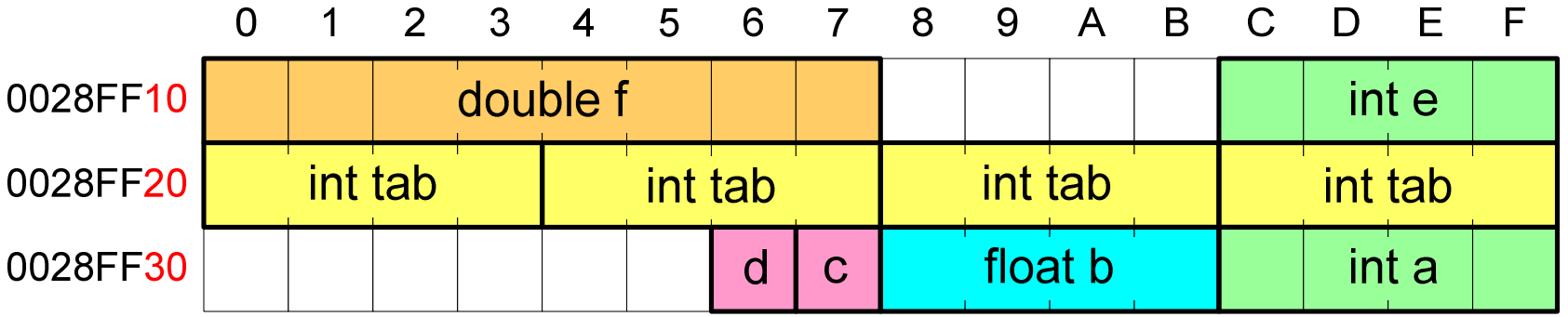
dr inż. Jarosław Forenc

Co to jest wskaźnik?

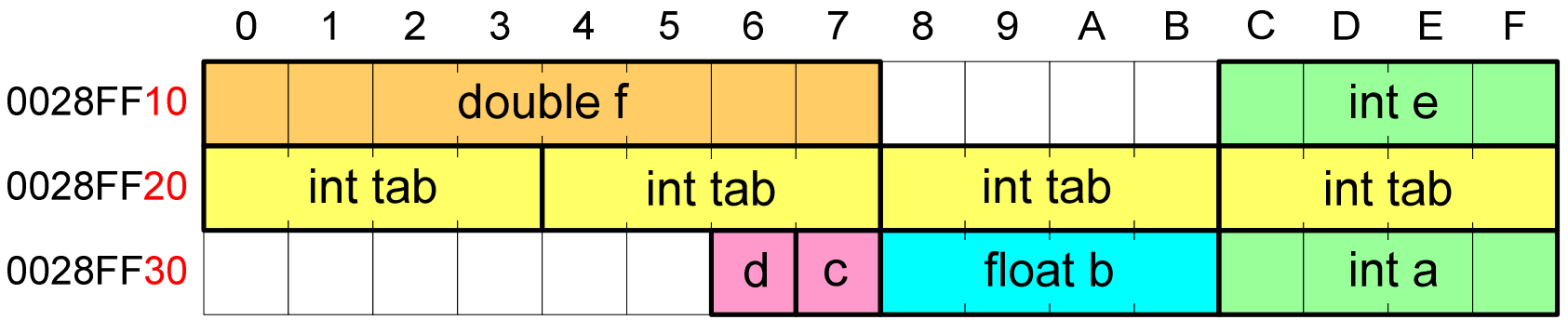
- Wskaźnik - zmienna mogącą zawierać adres obszaru pamięci - najczęściej adres innej zmiennej (obiektu)

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

- Zmienne przechowywane są w pamięci komputera



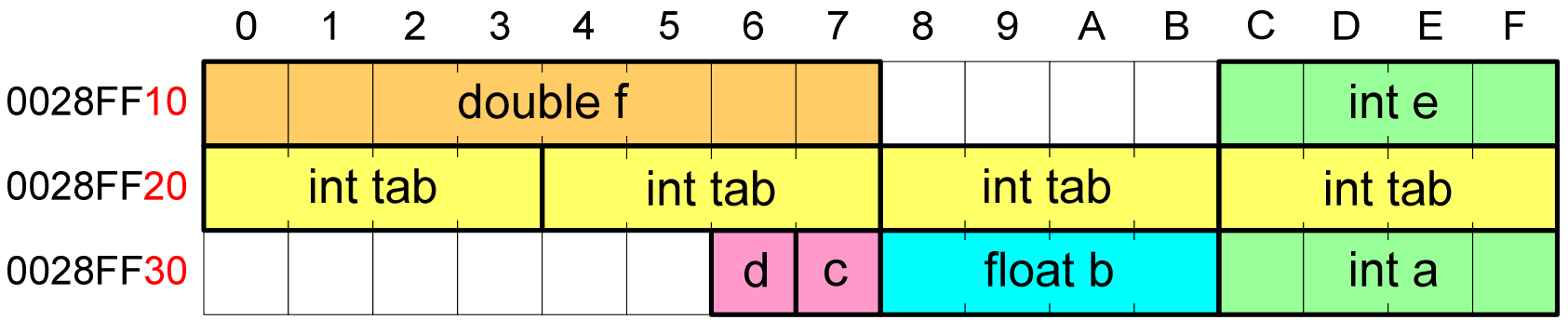
Co to jest wskaźnik?



- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji wszystkie nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
cout << "Adres zmiennej a:   " << &a << endl;
cout << "Adres tablicy tab: " << tab << endl;
```

Co to jest wskaźnik?



- Każda zmienna znajduje się pod konkretnym adresem i zależnie od typu zajmuje określoną liczbę bajtów
- Podczas kompilacji wszystkie nazwy zmiennych zastępowane są ich adresami
- Wyświetlenie adresu zmiennej:

```
Adres zmiennej a:  0028FF3C
Adres tablicy tab: 0028FF20
```

```
cout << "Adres zmiennej a.      << &a << endl;
cout << "Adres tablicy tab: " << tab << endl;
```

Deklaracja wskaźnika

- Deklarując wskaźnik (zmienną wskazującą) należy podać typ obiektu na jaki on wskazuje, a jego nazwę poprzedzić symbolem gwiazdki (*)

`typ *nazwa;`	`typ* nazwa;`	`typ * nazwa;`	`typ*nazwa;`

- Deklaracja zmiennej wskaźnikowej do typu int

```
int *ptr;
```

- Mówimy, że zmienna ptr jest typu: wskaźnik do zmiennej typu int
- Do przechowywania adresu zmiennej typu double trzeba zadeklarować zmienną typu: wskaźnik do zmiennej typu double

```
double *ptrd;
```

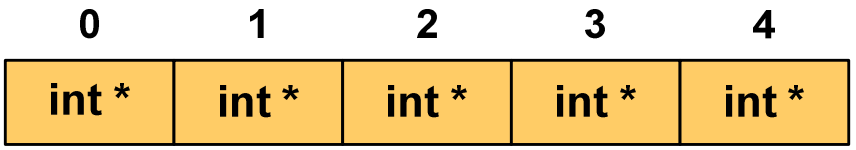
Deklaracja wskaźnika

- Można konstruować wskaźniki do danych dowolnego typu łącznie z typami wskaźnik do wskaźnika do...

```
char **wsk;
```

- Można deklarować tablice wskaźników - zmienna tab_ptr jest tablicą zawierającą 5 wskaźników do typu int

```
int *tab_ptr[5];
```



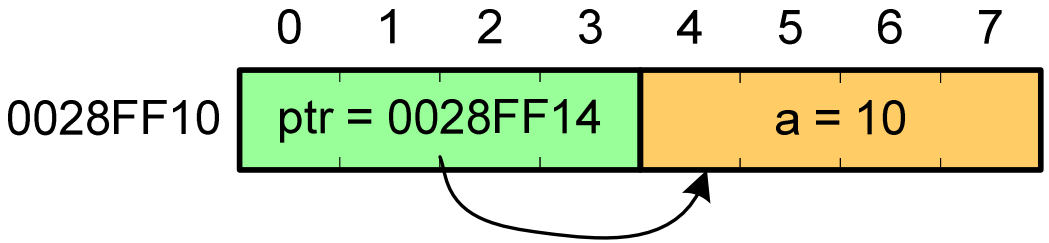
- Natomiast zmienna ptr_tab jest wskaźnikiem do 5-elementowej tablicy liczb int

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

Przypisywanie wartości wskaźnikom

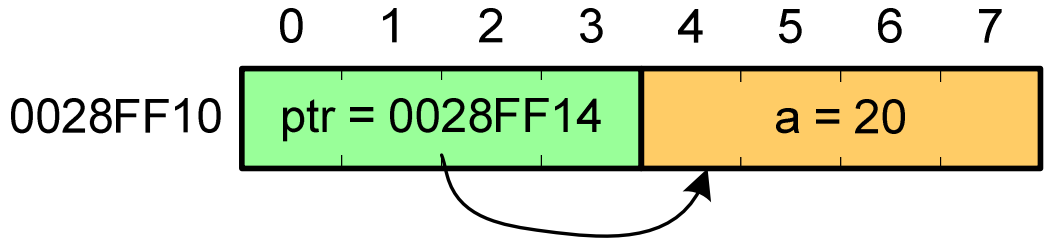
- Wskaźnikom można przypisywać adresy zmiennych
- Adresy takie tworzy się za pomocą operatora pobierania adresu &

```
int a = 10, *ptr;
ptr = &a;
```



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

```
*ptr = 20;
```



- Wskaźnik pusty:

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

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

Przykład: przypisywanie wartości wskaźnikom

```
#include <iostream>
using namespace std;

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

    cout << "x =    " << x << endl;
    cout << "ptri = " << ptri << endl;

    ptri = &x;              // przypisanie adresu
    cout << "ptri = " << ptri << endl;

    *ptri = *ptri + 10;   // x = x + 10
    cout << "x =    " << x << endl;
    cout << "x =    " << *ptri << endl;
}
```

```
x =    15
ptri = 0000000000000000
ptri = 000000000010FF960
x =    25
x =    25
```

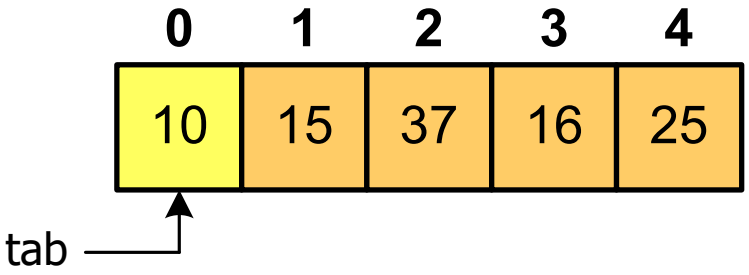
Arytmetyka wskaźnikowa

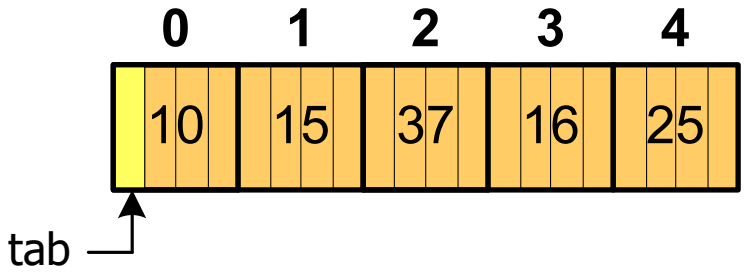
- Dodanie liczby całkowitej do wskaźnika - przed dodaniem liczby całkowitej jest ona mnożona przez liczbę bajtów zajmowanych przez wartość wskazywanego typu
- Zwiększenie wskaźnika (inkrementacja) - do wskaźnika można dodać 1 lub zastosować operator ++
- Odjęcie liczby całkowitej od wskaźnika - działa analogicznie jak dodanie liczby całkowitej do wskaźnika, ale wskaźnik musi być lewym operandem odejmowania
- Zmniejszenie wskaźnika (dekrementacja) - działa analogicznie jak inkrementacja

Wskaźniki a tablice

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

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



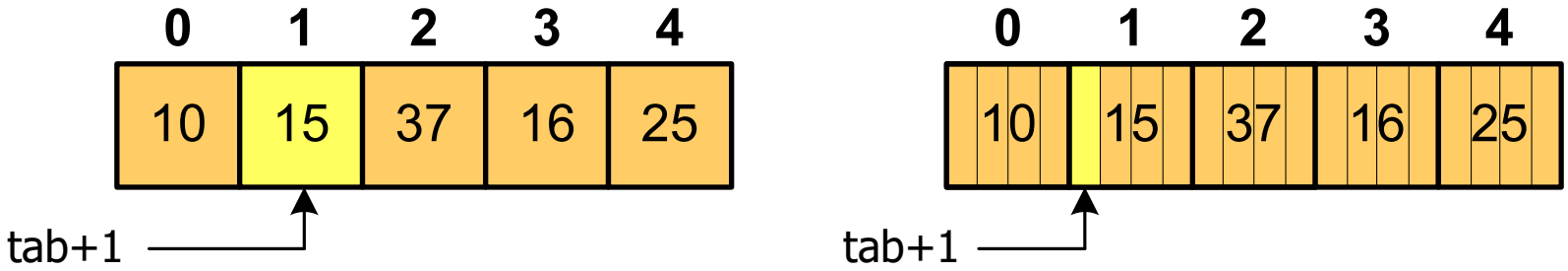


- Zastosowanie operatora * przed nazwą tablicy pozwala „dostać się” do zawartości elementu o indeksie 0

*tab jest równoważne tab[0]

Wskaźniki a tablice

- Dodanie 1 do adresu tablicy przenosi nas do elementu tablicy o indeksie 1 (przesunięcie o 4 bajty, gdyż int zajmuje tyle bajtów)



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

x = *tab+1; jest równoważne x = tab[0]+1;

Przykład: operacje na wskaźnikach

```
int tab[5] = {0,2,4,6,8}, *ptr;

cout << "Adres tab:      " << tab << endl;
cout << "Adres tab[0]:   " << &tab[0] << endl;
cout << "Adres tab+2:    " << tab+2 << endl;
cout << "Adres tab[2]:   " << &tab[2] << endl;
cout << "Wartosc tab:    " << *tab << endl;
cout << "Wartosc tab+2   " << *(tab+2) << endl;
```

```
Adres tab:      00000000012FFCC0
Adres tab[0]:   00000000012FFCC0
Adres tab+2:    00000000012FFCC8
Adres tab[2]:   00000000012FFCC8
Wartosc tab:    0
Wartosc tab+2   4
```

Przykład: operacje na wskaźnikach

```
int tab[5] = {0,2,4,6,8}, *ptr;

ptr = tab;
cout << "Adres ptr:      " << ptr << endl;
cout << "Wartosc *ptr:   " << *ptr << endl;
ptr++;
cout << "Adres ptr:      " << ptr << endl;
cout << "Wartosc *ptr:   " << *ptr << endl;
ptr = ptr + 2;
cout << "Adres ptr:      " << ptr << endl;
cout << "Wartosc *ptr:   " << *ptr << endl;
```

```
Adres ptr:      00000000012FFCC0
Wartosc *ptr:   0
Adres ptr:      00000000012FFCC4
Wartosc *ptr:   2
Adres ptr:      00000000012FFCCC
Wartosc *ptr:   6
```

Przykład: operacje na wskaźnikach

```
int tab[5] = {0,2,4,6,8}, *ptr;

ptr--;
cout << "Adres ptr:      " << ptr << endl;
cout << "Wartosc *ptr:   " << *ptr << endl;
ptr = ptr - 2;
cout << "Adres ptr:      " << ptr << endl;
cout << "Wartosc *ptr:   " << *ptr << endl;
```

```
Adres ptr:      00000000012FFCC8
Wartosc *ptr:   4
Adres ptr:      00000000012FFCC0
Wartosc *ptr:   0
```

Dynamiczny przydział pamięci

- Język C
  - przydział pamięci - funkcje malloc() i calloc()
  - zwolnienie pamięci - funkcja free()
- Język C++
  - przydział pamięci - operator new
  - zwolnienie pamięci - operator delete
- Operator new alokuje obszar pamięci niezbędny do przechowywania obiektu podanego typu i zwraca wskaźnik na początek tego obszaru
- Jeśli alokacja pamięci nie jest możliwa, to zwracana wartość NULL

Przykład - przydział pamięci na jedną zmienną

```
wartosc = 123.45
```

```
#include <iostream>
using namespace std;

int main()
{
    float *wsk;

    wsk = new float;
    if (wsk == NULL)
    {
        cout << "Blad przydzialu pamieci" << endl;
        return 0;
    }

    *wsk = 123.45f;
    cout << "wartosc = " << *wsk << endl;

    delete wsk;
}
```

Przykład - przydział pamięci na tablicę

```
#include <iostream>
using namespace std;

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

    tab = new int[n];

    for (int i=0; i<n; i++)
    {
        tab[i] = i*i;
        cout << "tab[" << i << "] = " << tab[i] << endl;
    }

    delete [] tab;
}
```

```
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
```

Przykład - przydział pamięci na strukturę

```
#include <iostream>
using namespace std;

struct punkt
{
   int x, y;
};

int main()
{
   punkt pkt, *wpkt;

   wpkt = new punkt;

   pkt.x = 10;   pkt.y = 20;
   wpkt->x = 30; wpkt->y = 40;
   cout << pkt.x << "  " << wpkt->x << endl;

   delete wpkt;
}
```

```
10  30
```

Program i funkcje

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

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

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

    d = a * sqrt(2.0f);
    cout << "Bok = " << a << ", przekatna = " << d;
}
```

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

Program i funkcje

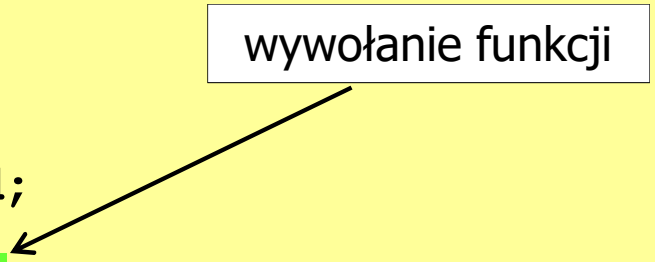
- Program składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

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

    d = a * sqrt(2.0f);
    cout << "Bok = " << a << ", przekatna = " << d;
}
```

Program i funkcje

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

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

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

    d = a * sqrt(2.0f);
    cout << "Bok = " << a << ", przekatna = " << d;
}
```



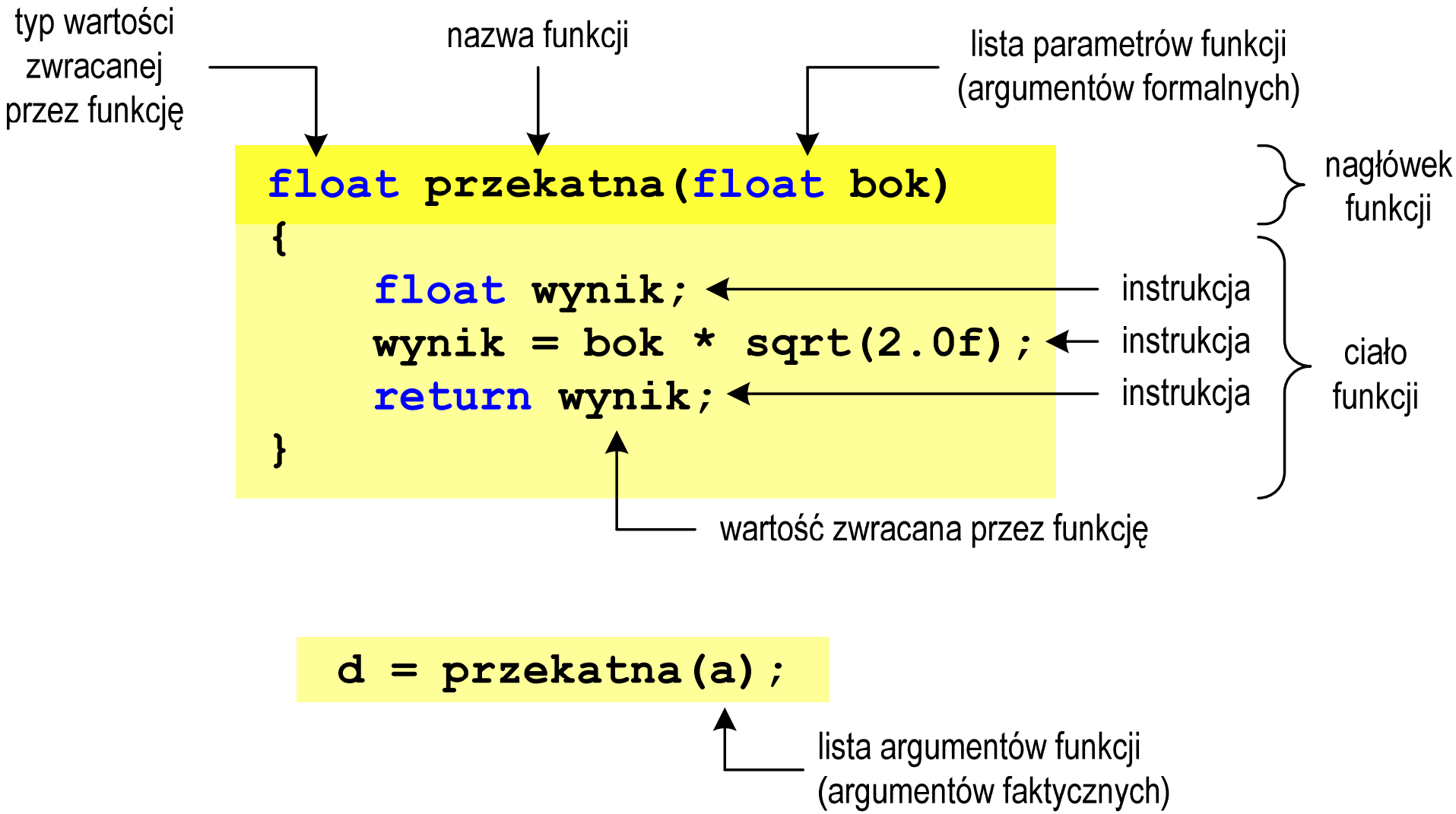
Program i funkcje

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

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

int main(void)
{
    float a = 10.0f, d;
    d = przekatna(a);
    cout << "Bok = " << a << ", przekatna = " << d;
}
```

definicja funkcji

definicja funkcji

Ogólna struktura funkcji




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

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

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

```
cout << "Bok = " << a << ", przekatna = ";
cout << przekatna(a) << endl;
```

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 <iostream>     /* przekątna prostokąta */
#include <cmath>
using namespace std;

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);
    cout << "Przekatna prostokata = " << d << endl;
}
```

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

Przeładowanie nazw funkcji

- Przeładowanie nazw funkcji polega na tym, że w danym zakresie ważności jest więcej niż jedna funkcja o tej samej nazwie:

```
int oblicz(int x);
int oblicz(int x, int y);
int oblicz(int x, double y);
```

- To, która funkcja zostanie w danym przypadku uaktywniona, zależy od liczby i typu argumentów
- Przy przeładowaniu ważna jest tylko odmienność listy argumentów, natomiast typ zwracany przez funkcję nie jest brany pod uwagę

```
int oblicz(int x);
float oblicz(int x);
```

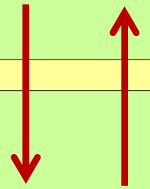
Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

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

int main(void)
{
    float a = 10.0f, d;
    d = przekatna(a);
    cout << "Bok = " << a << ", przekatna = " << d;
}
```

definicja funkcji

definicja funkcji



Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

int main(void)                                          definicja funkcji
{
    float a = 10.0f, d;
    d = przekatna(a);
    cout << "Bok = " << a << ", przekatna = " << d;
}

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

Prototyp funkcji

- Czy można zmienić kolejność definicji funkcji w kodzie programu?

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

int main(void)
{
    float a = 10.0f, d;
    d = przekatna(a);
    cout << "Bok = " << a << ", przekatna = " << d;
}

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

definicja funkcji

error C3861: 'przekatna': identifier not found

Prototyp funkcji

```
#include <iostream>     /* przekątna kwadratu */
#include <cmath>
using namespace std;

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

prototyp funkcji

```
int main(void)
{
    float a = 10.0f, d;
    d = przekatna(a);
    cout << "Bok = " << a << ", przekatna = " << d;
}
```

definicja funkcji

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

definicja funkcji

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);
```

Przekazywanie argumentów przez wartość

- W funkcji swap1 tworzone są lokalne kopie zmiennych skojarzonych z jej argumentami (zmienne a i b z funkcji main nie zmieniają swoich wartości)

```
a = 10, b = 20
```

```
#include <iostream>
using namespace std;

void swap1(int a, int b)
{
      int tmp;
      tmp = a; a = b; b = tmp;
}

int main(void)
{
      int a = 10, b = 20;

      swap1(a,b);
      cout << "a = " << a << ", b = " << b << endl;
}
```

Przekazywanie argumentów przez wskaźnik

- Do funkcji swap2 przekazywane są adresy zmiennych będących jej argumentami (zmienne a i b z funkcji main zmieniają swoje wartości)

```
a = 20, b = 10
```

```
#include <iostream>
using namespace std;

void swap2(int *a, int *b)
{
      int tmp;
      tmp = *a; *a = *b; *b = tmp;
}

int main(void)
{
      int a = 10, b = 20;

      swap2(&a,&b);
      cout << "a = " << a << ", b = " << b << endl;
}
```

Przekazywanie argumentów przez referencję

- Zmienne a i b z funkcji main zmieniają swoje wartości (ten sposób nie jest zalecany, gdyż program jest trudny do analizy)

```
#include <iostream>
using namespace std;

void swap3(int &a, int &b)
{
      int tmp;
      tmp = a; a = b; b = tmp;
}

int main(void)
{
      int a = 10, b = 20;

      swap3(a,b);
      cout << "a = " << a << ", b = " << b << endl;
}
```

```
a = 20, b = 10
```