Fundusze Europejskie Wiedza Edukacja Rozwój | Rzeczpospolita Polska | Unia Europejska Europejski Fundusz Społeczny

Wydział Elektryczny
Katedra Elektrotechniki Teoretycznej i Metrologii

Materiały do wykładu z przedmiotu:
Informatyka
Kod: EDS1B1007

WYKŁAD NR 8

Opracował: dr inż. Jarosław Forenc

Białystok 2019

Materiały zostały opracowane w ramach projektu „PB2020 - Zintegrowany Program Rozwoju Politechniki Białostockiej" realizowanego w ramach Działania 3.5 Programu Operacyjnego Wiedza, Edukacja, Rozwój 2014-2020 współfinansowanego ze środków Europejskiego Funduszu Społecznego.

Plan wykładu nr 8

- Dynamiczny przydział pamięci
- Funkcje w języku C
- Prototypy funkcji, typy funkcji

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 (np. największy rozmiar tablicy elementów typu char w języku C wynosi ok. 1 000 000)
- 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);
```


Dynamiczny przydział pamięci na wektor

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

int main(void)
{
   int   *tab, i, n, x;
   float suma = 0.0;

   printf("Podaj ilosc liczb: ");
   scanf("%d",&n);

   tab = (int *) calloc(n,sizeof(int));
   if (tab == NULL)
   {
      printf("Nie mozna przydzielic pamieci.\n");
      exit(-1);
   }
```


Dynamiczny przydział pamięci na wektor

```
   for (i=0; i<n; i++)   /* wczytanie liczb */
   {
      printf("Podaj liczbe nr %d: ",i+1);
      scanf("%d",&x);
      tab[i] = x;
   }

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

   printf("Srednia %d liczb wynosi %f\n",n,suma/n);

   free(tab);

   return 0;
}
```

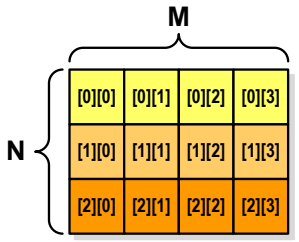
Dynamiczny przydział pamięci na wektor

```
    for (i=0; i<n; i++)
    {
       printf("Podaj lic
       scanf("%d",&x);
       tab[i] = x;
    }

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

    printf("Srednia %d liczb wynosi %f\n",n,suma/n);

    free(tab);

    return 0;
}
```

```
Podaj ilosc liczb: 5
Podaj liczbe nr 1: 1
Podaj liczbe nr 2: 2
Podaj liczbe nr 3: 3
Podaj liczbe nr 4: 4
Podaj liczbe nr 5: 5
Srednia 5 liczb wynosi 3.000000
```


Dynamiczny przydział pamięci na macierz

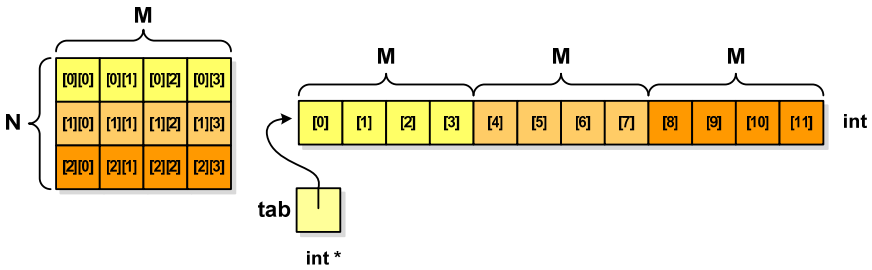
- Funkcje calloc() i malloc() umożliwiają bezpośrednio przydział pamięci tylko na wektor elementów
- Dynamiczny przydział pamięci na macierz wymaga zastosowania specjalnych metod
- Przydzielamy pamięć na macierz zawierającą N-wierszy i M-kolumn



Dynamiczny przydział pamięci na macierz (1)

- Wektor N×M-elementowy
- Przydział pamięci:

```
int *tab = (int *) calloc(N*M,sizeof(int));
```



Dynamiczny przydział pamięci na macierz (1)

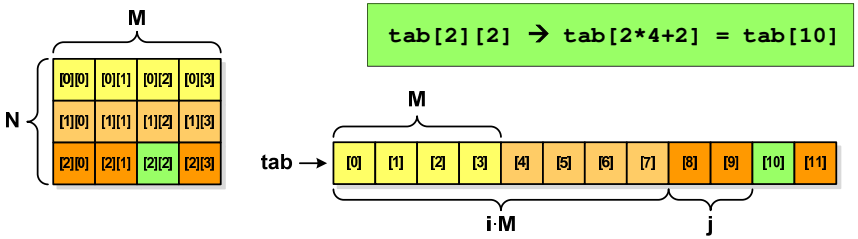
- Odwołanie do elementów macierzy:

```
tab[i*M+j]
```

lub

```
*(tab+i*M+j)
```

```
tab[2][2] → tab[2*4+2] = tab[10]
```



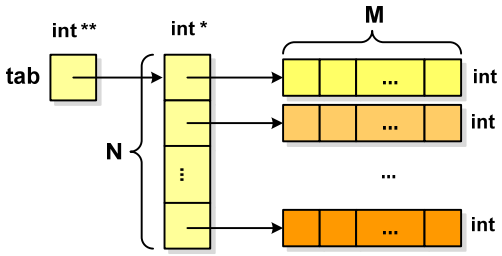
- Zwolnienie pamięci:

```
free(tab);
```

Dynamiczny przydział pamięci na macierz (2)

- N-elementowy wektor wskaźników + N-wektorów M-elementowych
- Przydział pamięci:

```
int **tab = (int **) calloc(N,sizeof(int *));
for (i=0; i<N; i++)
    tab[i] = (int *) calloc(M,sizeof(int));
```

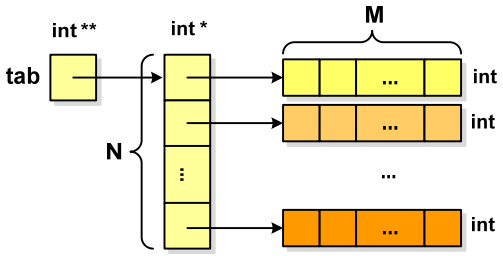


Dynamiczny przydział pamięci na macierz (2)

- Odwołania do elementów macierzy:
- Zwolnienie pamięci:

```
tab[i][j]
```

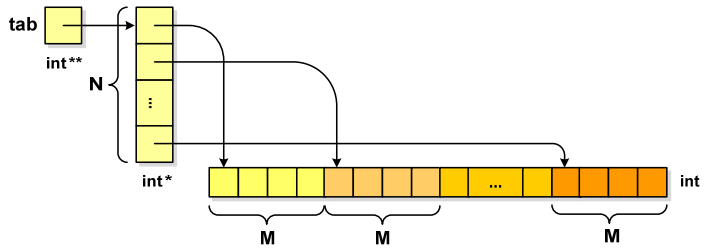
```
for (i=0; i<N; i++)
    free(tab[i]);
free(tab);
```



Dynamiczny przydział pamięci na macierz (3)

- N-elementowy wektor wskaźników + wektor N×M-elementowy
- Przydział pamięci:

```
int **tab = (int **) malloc(N*sizeof(int *));
tab[0] = (int *) malloc(N*M*sizeof(int));
for (i=1; i<N; i++)
   tab[i] = tab[0]+i*M;
```

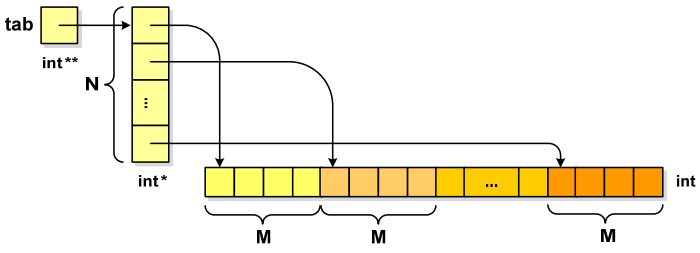


Dynamiczny przydział pamięci na macierz (3)

- Odwołania do elementów macierzy:
- Zwolnienie pamięci:

```
tab[i][j]
```

```
free(tab[0]);
free(tab);
```



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)                          definicja funkcji
{
    float a = 10.0f, d;

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

    return 0;
}
```


Program w języku C

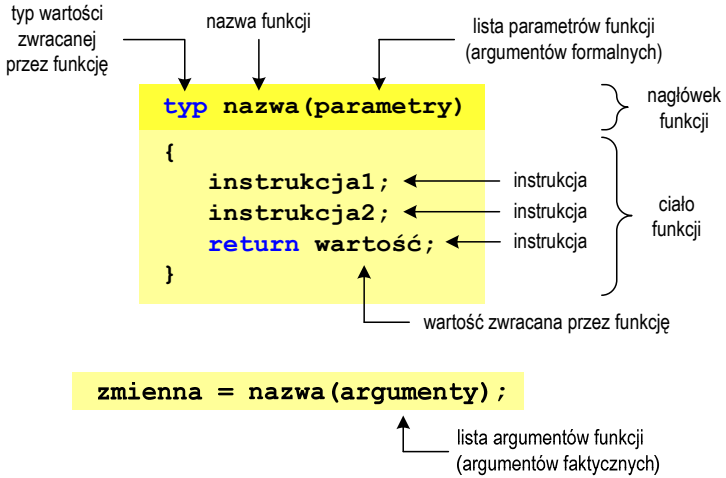
- Program w języku C składa się z funkcji i zmiennych
  - funkcje zawierają instrukcje wykonujące operacje
  - zmienne przechowują wartości

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

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

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

    return 0;
}
```

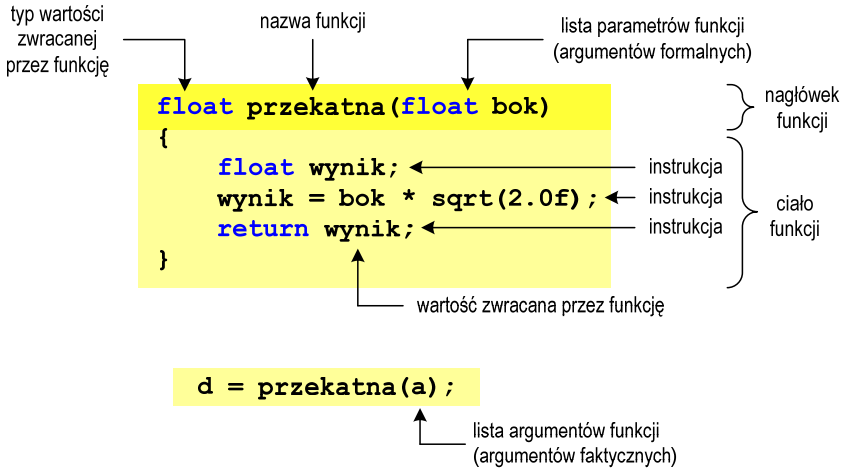

Funkcje w języku C

```
#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;
}
```

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[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",2019-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:           24
Rok urodzenia:  1995

Imie:           Barbara
Nazwisko:       Nowak
Wiek:           29
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;
}
```


Koniec wykładu nr 8

Dziękuję za uwagę!