Środowisko programisty + Języki Programowania

Wykład 5

dr Maciej Dziemiańczuk

Instytut Informatyki,
Wydział Matematyki, Fizyki i Informatyki, Uniwersytet Gdański

Rok akademicki 2022/2023

Zanim zaczniemy...

Powtórka

Co było na ostatnim wykładzie?

- instrukcja `switch`,
- pętla `while`, `do-while`, `for`
- tablice zmiennych

Co nas dzisiaj czeka?

- funkcje
- zmienne lokalne, globalne, statyczne
- tablice znaków c.d.
- funkcje na łańcuchach znaków

Funkcje

Funkcja

Wydzielony blok instrukcji w programie o określonej nazwie, typie oraz argumentach.

Składnia

```
typ nazwa ( argument/argumenty )
{
    instrukcje;
}
```

Kilka podstawowych własności

- Typem funkcji może być dowolny typ danych (np. `int`, `char`, `float` itd.)
- Typem funkcji, która nic nie zwraca jest `void`.
- Argument funkcji to typ oraz nazwa (np. `int n`).
- Jeśli funkcja nie ma argumentów, wpisujemy w to miejsce `void`.
- Jeśli funkcja ma kilka argumentów, to są one oddzielone przecinkami.

Wywołanie funkcji

```
nazwa ( argument/argumenty );
```

Przykłady funkcji

(a) Funkcja `Powitanie()`, która nic nie zwraca i nie ma argumentów

Kod programu: fun1.c

```
1  #include <stdio.h>
2
3  // definicja funkcji
4  void Powitanie( void )
5  {
6      printf("Witaj ");
7      printf("uzytkowniku\n");
8  }
9
10 int main( void )
11 {
12     // Wywolanie funkcji
13     Powitanie();
14     Powitanie();
15 }
```

Pytanie: *Czy funkcja może wywoływać inną funkcję?*

Przykłady funkcji

(b) Funkcja `Kwadrat()`, która nic nie zwraca i ma jeden argument typu `int`

Kod programu: fun2.c

```
1   #include <stdio.h>
2
3   void Kwadrat( int a )
4   {
5       int wynik = a*a;
6       printf("Kwadrat liczby %i to %i \n", a, wynik);
7   }
8
9   int main( void )
10  {
11      Kwadrat(10);
12
13      int liczba = 5;
14      Kwadrat(liczba);
15      Kwadrat(liczba*2);
16  }
```

Pytanie: *Co łączy zmienne `liczba` oraz `a`? Które zmienią swoją wartość i dlaczego?*

Przykłady funkcji

(c) Funkcja `Suma()`, która nic nie zwraca i ma dwa argumenty oba typu `int`

Kod programu: fun3.c

```
1   #include <stdio.h>
2
3   void Suma( int a, int b )
4   {
5       if (a > b) return;
6
7       int suma = 0;
8       while (a <= b) {
9           suma += a++;
10      }
11      printf("Suma wynosi %i\n", suma);
12  }
13
14  int main( void )
15  {
16      Suma(0, 10);
17  }
```

Pytanie: *Co właściwie robi instrukcja* ***return****?*

Przykłady funkcji

(d) Funkcja `Kwa()`, które zwraca typ `int` i ma jeden argument typu `int`.

Uwaga! *Do zwracania wartości przez funkcję używamy operatora* ***return****, który jednocześnie przerywa działanie funkcji w tym miejscu.*

Kod programu: fun4.c

```
1   #include <stdio.h>
2
3   int Kwa( int a )
4   {
5       int k = a * a;
6       return k;
7   }
8
9   int main( void )
10  {
11      int a, b = 10;
12
13      // Wywolanie funkcji ktora zwraca int'a
14      a = Kwa(b);
15
16      print("%i\n", a);
17  }
```

Pytanie: *Co robi funkcja* ***Kwa()****?*

Przykłady funkcji

(e) Funkcja `suma()`, która przyjmuje zmienną tablicową oraz jej rozmiar

Kod programu: fun5.c

```
1  #include <stdio.h>
2
3  int suma(int lista[], int rozmiar)
4  {
5      int suma = 0;
6      while (rozmiar-- > 0)
7          suma += lista[rozmiar];
8      return  suma;
9  }
10
11 int main( void )
12 {
13     int liczby[] = {5, 1, 8, 10};
14     printf("wartosc %i \n", suma(liczby, 4));
15 }
```

Uwaga! *Podczas przekazywania tablicy pamiętajmy również o przekazaniu jej rozmiaru. Operator* ***sizeof*** *użyty do tablicy przekazanej jako argument do funkcji nie zwróci jej rozmiaru.*

Wywołanie funkcji w innej funkcji

Kod programu: funfun.c

```
1   #include <stdio.h>
2
3   int Abs(int a)
4   {
5       if (a < 0) return -a; else return a;
6   }
7
8   int Dist( int a, int b )
9   {
10      return Abs(a - b);
11  }
12
13  int main( void )
14  {
15      int x = -10;
16      int y = 20;
17
18      printf("Odleglosc wynosi : %i \n", Dist(x, y));
19  }
```

Pytanie: *Czy możemy zamienić kolejność kodu funkcji* ***Abs*** *oraz* ***Dist*** *w programie?*

Co jest nie tak w poniższym programie?

Kod programu: funloop.c

```c
#include <stdio.h>

float pierwsza(int n, float x)
{
    if (n == 0) return x;
    return n * druga(n-1, x);
}

float druga(int n, float x)
{
    if (n == 0) return 1.0;
    return 2.0f * pierwsza(n-1, x/2.0f);
}

int main()
{
    float liczba = pierwsza(10, 0.5);
    printf("%f \n", liczba);
}
```

Pytanie: *W jaki sposób naprawić ten program?*

Deklaracja funkcji a jej implementacja

Deklaracja (prototyp) funkcji – informacja o nazwie, typie zwracanym oraz argumentach

```
int nazwa_funkcji(int arg1, float arg2);
int nazwa_funkcji(int, float);       // nie trzeba podawac nazw zmiennych
```

Implementacja funkcji (definicja funkcji) – deklaracja oraz ciało funkcji

```
int nazwa_funkcji(int arg1, float arg2)
{
   instrukcje; // cialo funkcji
}
```

Pliki nagłówkowe `.h` zawierają deklaracje funkcji

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

Korzystając z oddzielnych deklaracji funkcji możemy pozbyć się problemu widoczności funkcji.

Deklaracja funkcji a jej implementacja

Przykładowy szablon programu z wieloma funkcjami

```
1   // Dolaczanie plikow naglowkowych
2   #include <stdio.h>
3
4   // Deklaracje (prototypy) funkcji
5   int fun1(int, float);
6   int fun2(double);
7
8   // Implementacje funkcji juz w dowolnej kolejnosci
9   int fun1(int a, float b) {
10      ...
11  }
12  int fun2(double n) {
13      ...
14  }
15
16  // Glowna funkcja
17  int main() {
18
19  }
```

Do zagadnienia wieloplikowych projektów wrócimy w przyszłości ...

A tymczasem naprawmy program z poprzedniego przykładu

Poprawiony kod z poprzedniego przykładu

Kod programu: funfinal.c

```c
#include <stdio.h>

// deklaracje
float pierwsza(int, float);
float druga(int, float);

// glowna funkcja
int main()
{
   float liczba = pierwsza(10, 0.5);
   printf("%f \n", liczba);
}

// dodatkowe funkcje
float pierwsza(int n, float x)
{
   if (n == 0) return x;
   return n * druga(n-1, x);
}

float druga(int n, float x)
{
   if (n == 0) return 1.0;
   return 2.0f * pierwsza(n-1, x/2.0f);
}
```

Funkcje rekurencyjne

Funkcja rekurencyjna

Funkcja, która wywołuje samą siebie. W takich funkcjach bardzo ważny jest warunek przerywający wywoływanie samej siebie.

```
void funName(...)
{
    ...
    funName(...)
    ...
}
```

Przykład funkcji rekurencyjnej

```
1   int Box(int n)
2   {
3       if (n <= 0) return 0;
4       int val = n + Box(n-1);
5       return val;
6   }
```

Pytanie: *Co zwróci wywołanie funkcji `Box(10)`?*

Funkcje rekurencyjne

Przykład bardzo znanej funkcji rekurencyjnej

Kod programu: fib.c

```c
#include <stdio.h>

int F(int n)
{
    if (n < 0)
        return 0;
    else if (n == 0 || n == 1)
        return 1;
    else
        return F(n-1) + F(n-2);
}

int main ( void )
{
    int liczba = F(10);
    printf("F(10) = %i\n", liczba);
}
```

Pytanie: *Dla jak dużego argumentu funkcji* ***F*** *uda się ją wywołać i doczekać końca?*

Funkcje rekurencyjne

(1) Funkcje rekurencyjne mają często prosty zapis, ale bez dodatkowych technik spamiętujących czas działania może być bardzo nieoptymalny.

```
int Fibonacci(int n)
{
    if (n < 0) return 0;
    else if (n == 0 || n == 1) return 1;
    else return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Pytanie: *Ile łącznie razy wywoła się funkcja* `Fibonacci` *dla* $n = 10$*?*

(2) Proces zamiany kodu rekurencyjnego na iteracyjny nazywamy derekursywacją

Zadanie: *Napisz program, który liczy n-tą liczbę Fibonacciego za pomocą pętli.*

Pytania dla ciekawskich:

- Czy każdy kod rekurencyjny można zamienić na iteracyjny?
- Co to jest rekurencja ogonowa?

... wracając do Wykładu 4

Zadanie
Napisz program, który wczytuje N liczb i wypisuje je w odwrotnej kolejności.

Rozwiązanie korzystające z pętli i tablic.
Pytanie z przyszłości: czy do tego problemu koniecznie trzeba użyć tablic i pętli?

Pytanie: *A gdyby tak użyć funkcji, która ...?*

Zadanie: *Rozwiąż powyższe zadania bez użycia tablic i pętli (oraz oczywiście deklarowania ręcznie N zmiennych).*

Funkcje - błędy

Uwaga na błędy:

- Ważna jest kolejność i ilość argumentów

```
1  void Suma(int a, int b)
2  {
3      printf("Suma wynosi %i \n", a + b );
4  }
5  ...
6  Suma(15);
```

- Brakujący operator `return`

```
1  int Iloczyn(int a, int b)
2  {
3      int wynik = a * b;
4  }
5  ...
6  int a = Iloczyn(5, 10);
```

Funkcje - błędy

Uwaga na błędy:

- Funkcja, która nic nie zwraca

```
1   void Kwadrat(int a)
2   {
3       int wynik = a*a;
4       printf("%i^2 = %i \n", a, wynik);
5   }
6   ...
7   int k = Kwadrat(8);
```

Zmienne lokalne, globalne, statyczne i rejestrowe

Rodzaje zmiennych

Zmienne możemy podzielić na kilka rodzajów, ze względu na:

- zasięg widoczności,
- czas życia,
- sposób przechowywania.

Podstawowe rodzaje zmiennych

(1) lokalne (automatyczne)

(2) globalne

(3) statyczne

(4) rejestrowe

Zmienne lokalne auto (automatyczne)

Zdefiniowane w bloku instrukcji i widoczne tylko w tym bloku oraz blokach wewnętrznych.

```
auto int x;
int x;       // klasyfikator auto jest opcjonalny
```

Przykład

Kod programu: zm-lok.c

```
1   #include <stdio.h>
2
3   int main( void )
4   {
5       int a = 10;
6       {
7           int b = 5;
8           printf("%i %i\n", a, b);
9       }
10
11      printf("%i %i\n", a, b);    // blad!
12  }
```

Uwaga! *Zmienne automatyczne, które nie zostały zainicjalizowane mają losową zawartość!*

Zmienne lokalne i funkcje

Zmienne lokalne nie są widziane pomiędzy funkcjami.

Przykład

Kod programu: zm-lok-fun.c

```
1   #include <stdio.h>
2
3   void kwadrat(int b)
4   {
5       auto int wynik = b * b; // zmienna lokalna w funkcji kwadrat
6       printf("%i\n", wynik);
7   }
8
9   int main( void )
10  {
11      int a = 10; // zmienna lokalna w funkcji main()
12      kwadrat(a);
13  }
```

Pytanie: *Czy jest jakiś związek pomiędzy zmienną* `b` *a zmienną* `a`*?*

Zmienne lokalne i funkcje

Argumenty funkcji to zmienne lokalne danej funkcji.

Przykład

Kod programu: zm-lok-fun-arg.c

```c
#include <stdio.h>

void zwieksz(int b) {
    b = b + 1;
}

int main( void )
{
    int a = 15;
    zwieksz(a);

    printf("a = %i\n", a);
}
```

Pytanie: *Co się wyświetli? Czy zmiana zmiennej `b` na `a` coś zmieni?*

Do problemu przekazywania argumentów przez wartość/adres wrócimy przy okazji wskaźników.

Zmienne globalne

Zmienne zdefiniowane poza funkcjami i widoczne we wszystkich funkcjach w dalszej części kodu i ewentualnie w innych plikach projektu. Deklarowane w lokalnym bloku przy użyciu operatora `extern` (opcjonalnie).

Przykład

Kod programu: zm-glob.c

```c
#include <stdio.h>

int YEAR = 2025;  // zmienna globalna

void sub(int n) {
    extern int YEAR; // opcjonalna deklaracja zmiennej jako globalna
    YEAR = YEAR - n;
}

int main( void )
{
    sub(1995);
    printf("AGE = %i\n", YEAR);
}
```

Uwaga! *Zmienne globalne, które nie zostały zainicjalizowane jawnie mają wartość zerową.*

Zmienne globalne

Mają zalety i wady, jakie?

Wady:

- utrudniają przenoszenie kodu do innych projektów (konflikt nazw zmiennych globalnych),
- utrudniają zrównoleglanie programów (konflikt dostępu do wspólnej pamięci).

Zalety:

- w prostych projektach ułatwiają komunikację pomiędzy funkcjami.

Zmienne globalne

Przykład możliwości poprawy kodu przy użyciu zmiennej globalnej

Kod programu: zm-glob1.c

```
1  #include <stdio.h>
2
3  void wypisz(int tab[], int rozmiar) {
4     for (int i=0; i<rozmiar; ++i) printf("%i ", tab[i]);
5     printf("\n");
6  }
7  void wygeneruj(int tab[], int rozmiar) {
8     tab[0] = tab[1] = 1;
9     for (int i=2; i<rozmiar; ++i) tab[i] = tab[i-1] + tab[i-2];
10 }
11 void kwadrat(int tab[], int rozmiar) {
12    for (int i=2; i<rozmiar; ++i) tab[i] *= tab[i];
13 }
14
15 int main( void ) {
16    int tablica[10];
17
18    wygeneruj(tablica, 10);
19    kwadrat(tablica, 10);
20    wypisz(tablica, 10);
21 }
```

Zmienne statyczne `static`

Zmienne zdefiniowane w bloku instrukcji i widoczne tylko w nim oraz w blokach wewnętrznych, ale których stan jest zapamiętywany pomiędzy wywołaniami tego bloku.

Przykład

Kod programu: zm-stat.c

```c
#include <stdio.h>

void funkcja()
{
    static int n = 0; // zmienna statyczna
    n = n + 1;
    printf("Wywoluje sie %i raz\n", n);
}

int main( void )
{
    funkcja();
    funkcja();
    funkcja();
}
```

Uwaga! *Zmienna statyczna, która nie jest zainicjalizowana jawnie ma wartość zero.*

Zmienne rejestrowe `register`

Zmienne zdefiniowane w bloku instrukcji i widoczne tylko w nim i blokach wewnętrznych, ale przechowywane w rejestrach procesora (o ile to możliwe).

```
register int x;
```

Przykład

Kod programu: zm-reg.c

```
1   #include <stdio.h>
2   void liczby(int n)
3   {
4       register int i = 0;        // zmienna rejestrowa
5       for (i = 0; i<n; ++i)
6           printf("i=%i\n", i);
7   }
8
9   int main( void )
10  {
11      liczby(2000);
12  }
```

Pytanie z przyszłości: *Czy zmienne rejestrowe mają tylko zalety?*

Określ rodzaj zmiennych i podaj wynik działania programu.

Kod programu: zm-zad.c

```c
#include <stdio.h>

int k = 20;

void funkcja(int k)
{
    {
        auto int k = 10;
        printf("%i\n", k);
    }
    {
        printf("%i\n", k);
    }
    {
        extern int k;
        printf("%i\n", k);
    }
    printf("%i\n", k);
}
int main( void )
{
    int k = 10;
    funkcja(k-5);
    printf("%i\n", k);
}
```

Co wypisze poniższy program?

Kod programu: zm-egz.c

```c
#include <stdio.h>

int k = 20;
void funkcja(int k)
{
    --k;
    {
        auto int k = 10;
        printf("%i\n", --k);
    }
    {
        printf("%i\n", k+=5);
    }
    {
        extern int k;
        printf("%i\n", k/3);
    }
    printf("%i\n", k+5);
}
int main( void )
{
    int k = 10;
    funkcja(k--);
    printf("%i\n", k);
}
```

Łańcuchy znaków c.d.

Łańcuchy znaków

Tablice zmiennych typu **char**, w których występuje znak końca łańcucha **'\0'**.

```
char napis[] = "to jest napis";
char imie[] = { 'A', 'l', 'b', 'e', 'r', 't', '\0' };

printf("napis: %s ", imie);     // wypisywanie
scanf("%s", napis);             // wczytywanie
```

Wypisywanie łańcucha znaków znak po znaku za pomocą pętli **while** i **for**

```
int i=0;

// za pomoca petli while
while (napis[i] != '\0') {
   printf("%c,", napis[i]);
   i++;
}
// za pomoca petli for
for (i=0; i<DLUGOSC; i++)
   printf("%c," napis[i]);
```

Pytanie: *Jak obliczyć długość łańcucha znaków?*

Biblioteka standardowa

Zestaw funkcji będących częścią języka C, które składają się m.in. z

- funkcji wejścia/wyjścia `<stdio.h>`
- funkcji matematyczne `<math.h>` oraz `<complex.h>`
- funkcji daty i czasu `<time.h>`
- obsługi lokalizacji `<locale.h>`
- alokacji pamięci i innych podstawowych funkcji `<stdlib.h>`
- funkcji na łańcuchach znaków `<string.h>`
- funkcji na znakach `<ctype.h>`

I wielu innych... Zobaczmy na przykład cppreference.com.

Funkcje na łańcuchach znaków

Zestaw funkcji ze standardowej biblioteki obsługujących łańcuchy znaków.

Zobacz np. cppreference.com.

Pliki nagłówkowe

```
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
```

Rodzaje funkcji na łańcuchach znaków

(a) klasyfikujące
(b) konwersji
(c) manipulujące łańcuchami znaków
(d) porównujące i wyszukujące

Biblioteka ctype.h

Wybrane funkcje na znakach z ctype.h:

- `isalnum(int ch)` – czy `ch` jest cyfrą lub literą
- `isalpha(int ch)` – czy `ch` jest literą
- `isdigit(int ch)` – czy `ch` jest cyfrą
- `islower(int ch)` – czy `ch` jest małą literą
- `isupper(int ch)` – czy `ch` jest wielką literą
- `tolower(int ch)` – zamienia `ch` na małą literę
- `toupper(int ch)` – zamienia `ch` na wielką literę

Przykład odfiltrowania liter i cyfr

```
// kod programu: filtr.c
char czysty[255], smieci[] = "T$o#?--=J~//e+@)??!s#tTa$jn%@#eH!/-++a':s>l>o";
int i_in = 0, i_out = 0;

while (smieci[i_in] != '\0')
{
   if ( isalnum(smieci[i_in]) ) czysty[i_out++] = smieci[i_in];
   i_in++;
}
czysty[i_out] = '\0';   // obowiazkowe zakonczenie lancucha
```

Biblioteka `stdlib.h`

Wybrane funkcje konwersji z `stdlib.h`:

- `atof(const char* str)` - zamienia `str` na liczbę zmiennoprzecinkową
- `atoi(const char* str)` - zamienia `str` na liczbę całkowitą `int`
- `atol(const char* str)` - zamienia `str` na liczbę całkowitą `long`
- `atoll(const char* str)` - zamienia `str` na liczbę całkowitą `long long`

Przykład zamiany łańcucha znaków na zmienną liczbową

```
int liczba;
liczba = atoi("1029");

printf("liczba = %i \n", liczba);
```

```
float num;
num = atof("-1029.523");

printf("num = %f \n", num);
```

Pytanie: *A jeśli podamy łańcuch, który nie przedstawia liczby?*

Przykład użycia funkcji konwertujących na nietypowych danych

Kod programu: konw.c

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

int main ( void )
{
    int a = 5;
    a = atoi(" -123junk");
    printf("a = %i\n", a);

    int b = 5;
    b = atoi("junk");
    printf("b = %i\n", b);

    int c = 5;
    c = atoi("8.123");
    printf("c = %i\n", c);
}
```

Co zostanie wyświetlone?

Biblioteka `string.h`

Wybrane funkcje z `string.h`:

- `strcpy(char *dest, const char *src)` - kopiuje łańcuch znaków z `src` do `dest`
- `strcat(char *dest, const char *src)` - dopisuje łańcuch z `src` do `dest`
- `strlen(const char *str)` - zwraca długość łańcucha `src`
- `strcmp(const char *a, const *b)` - porównuje dwa łańcuchy znaków

Przykład wyliczenia długości łańcucha znaków za pomocą `strlen()`

```
// kod programu: str-len.c
char buffer[50];
int dlugosc = 0;

printf("Podaj napis: ");
scanf("%s", buffer);

dlugosc = strlen(buffer);
```

Pytanie: *Czy moglibyśmy sami policzyć długość łańcucha znaków?*

Porównanie dwóch łańcuchów znaków

Uwaga! *Poniższe rozwiązanie nie zadziała!*

```
1   char a[] = "napis1";
2   char b[] = "napis2";
3
4   if (a == b) ... // porownamy adresy obu tablic, a nie ich zawartosc
```

Pytanie: *Jak zatem porównać dwa napisy?*

Należy porównać oba łańcuchy znak po znaku lub wykorzystać funkcję `strcmp()`

Przykład

```
// Kod programu: str-cmp.c
char haslo[] = "T@jn3H@$10";
char buffer[] = "....";

if (strcmp(haslo, buffer) == 0)
   printf("Witaj w systemie!\n");
else
   printf("Witaj intruzie!\n");
```

Zauważ co funkcja ***strcmp*** *zwraca w przypadku, gdy oba napisy są sobie równe.*

Pytania?

Dodatkowe slajdy

Polecenie `fflush()`

Polecenie `fflush`

```
int fflush ( FILE * stream );
```

Dla przykładu `fflush(stdout)` wymusza wypisanie ewentualnych pozostałości w buforze standardowego wyjścia (na ekranie), natomiast `fflush(stdin)` opróżnia bufor standardowego wejścia (np. klawiatury) często używany ze `scanf`.

Przykładowe użycie

Kod programu: flush.c

```
1  #include <stdio.h>
2
3  int main ()
4  {
5      int liczba;
6      do
7      {
8          printf("Podaj liczbe od 1 do 10:");
9          scanf("%i", &liczba);    // sprobuj podac cos co nie jest liczba
10         // fflush(stdin);        // a nastepnie odkomentuj ta linie
11     } while (liczba < 1 || liczba > 10);
12
13     printf("Liczba %i\n", liczba);
14     return 0;
15 }
```

Polecenie `getch()` – poza standardową biblioteką

Polecenie `getch` - pobranie znaku ze standardowego wejścia bez potrzeby akceptacji klawiszem ENTER

```
// pliki nagłówkowe:
#include <conio.h> // windows
#include <ncurses.h> // linux
#include <termios.h> // linux

int znak = getch();
```

Przykładowe użycie (Windows)

Kod programu: getch.c

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  int main( void )
5  {
6      char c;
7      while ((c = getch()) != '.')
8          printf("%i\n",c);
9  }
```

Wersja dla linuxa. Alternatywa dla polecenia `getch()` ?

Pytania?