Języki i paradygmaty programowania 1 – studia stacjonarne 2025/26

Lab 3. Funkcje, argumenty funkcji, wskaźniki, adresy.

1. Podstawowe pojęcia:

   Funkcja *– w C (czasami nazywana podprogramem, rzadziej procedurą) to wydzielona część programu, która przetwarza argumenty i ewentualnie zwraca wartość, która następnie może być wykorzystana, jako argument w innych działaniach lub funkcjach. Funkcja może posiadać własne zmienne lokalne. Funkcje w C mogą zwracać dla tych samych argumentów różne wartości.*

   Definicja funkcji:

```
typ_zwracanej_wartości  nazwa_funkcji(lista deklaracji argumentów)
   {
          …ciało(treść) funkcji
          return Val; //zwracana wartość
   }
```

   - Funkcja przed użyciem musi być zadeklarowana/zdefiniowana;
   - Typ zmiennej **Val** musi być zgodny z **typ_zwracanej_wartości**;
   - Typy argumentów formalnych w definicji i deklaracji funkcji muszą być zgodne z typami argumentów aktualnych (w miejscu wywołania funkcji);
   - Typ ***void*** – przekazywany jako argument funkcji oznacza, że funkcja nie pobiera żadnych argumentów.

   Pojęcia:

   a. ***Parametr*** (argument formalny, parametr formalny) – to zmienna z listy parametrów w definicji funkcji)

   b. ***Argument*** (argument aktualny, parametr aktualny) – to zmienna (wartość) podawana podczas wywołania funkcji – to, co podstawia użytkownik

   c. ***Return*** – to słowo kluczowe języka C. W przypadku funkcji służy do:
   - Przerwania funkcji (i przejścia do następnej instrukcji w funkcji wywołującej)
   - Zwrócenia wartości

2. Przykład: funkcja ***power*** podnosi liczbę rzeczywistą x do potęgi całkowitej n.

```
#include <stdio.h>
double power(double base, int n); //deklaracja funkcji (prototyp)
int main()
{
      int pow = 15;
      double a = 35.0;
      double res = 0.0;
      res = power(a, pow); //wywołanie funkcji
      printf("a = %le pow = %d a**pow = %le\n", a, pow, res);
      return 0;
}
double power(double base, int n) //definicja funkcji
{
      int i;
      double res = 1.0;
      for (i = 1; i <= n; i++)
            {
                  res = res * base;
            }
      return res;
}
```

3. Funkcja nie musi zwracać żadnej wartości.

```
void nazwa_funkcji(lista deklaracji argumentów)
{
        // instrukcje
        return; //nie koniecznie
}
```

Należy jednak pamiętać, że może zdarzyć się taka sytuacja:

```
void nazwa_funkcji(lista deklaracji argumentów)
{
        // instrukcje
        if (wyrażenie_logiczne)
                return;
        // dalszy ciąg instrukcji
        return; //nie koniecznie
}
```

4. **Przekazywanie argumentów do funkcji przez wartość**.

Przekazywanie argumentów przez wartość - wywołując funkcję, wartości argumentów, z którymi ją wywołujemy, są kopiowane do funkcji. Oznacza to, że wewnątrz funkcji operujemy tylko na ich kopiach (tworzonych na stosie). Po zakończeniu działania funkcji, kopie te zostają zniszczone (usunięte ze stosu), a wszystkie zmiany wykonane na wartościach kopii przekazanych argumentów stracone.

Przykład. Napisz program obliczający wartości kilku wyrażeń arytmetycznych dla danej wartości y.

$$\alpha = \frac{1}{y^2 + \sqrt{1 + 2y + 3y^2}}$$

$$\beta = \frac{6.4 + y}{y^2 + \sqrt{1 + 2y + 3y^2}}$$

$$\gamma = \frac{\sin(y)}{y^4 + \sqrt{1 + 2y^2 + 3y^4}}$$

$$\delta = \frac{1}{\sin^2 y + \sqrt{1 + 2\sin(y) + 3\sin^2 y}}$$

Jeżeli przyjmiemy, że

$$f(x) = \frac{1}{x^2 + \sqrt{1 + 2x + 3x^2}}$$

To powyższe równania można zapisać krócej

$$\alpha = f(y)$$

$$\beta = (6.4 + y) f(y)$$

$$\gamma = f(y^2) \sin y$$

$$\delta = f(\sin y)$$

Przyjmując, że zmienna **i** jest typu **int**, dodatkowo policz wartość wyrażenia:

$$\varepsilon = 3.2 f(i)$$

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
        int i;
        double y, al, be, ga, de, ep;
        double ff(double);
```

```
        printf("Podaj liczbe rzeczywista ");
        scanf("%lf", &y);
        printf("Podaj liczbe calkowita ");
        scanf("%d", &i);
        al = ff(y);
        be = (6.4 + y) * ff(y);
        ga = sin(y) * ff(y*y);
        de = ff(sin(y));
        ep = ff((double)i)*3.2;
        printf("Wyniki: %lf %lf %lf %lf %lf\n", al, be, ga, de, ep);
        //  system("pause");
}
double ff(double x)
{
        double gg;
        gg = 1 / (x*x + sqrt(1 + 2 * x + 3 * x*x));
        return gg;
}
```

5. **Przekazywanie argumentów do funkcji przez wskaźnik.**

Wskaźnik *- to specjalny rodzaj zmiennej, w której zapisany jest adres w pamięci komputera. Oznacza to, że wskaźnik wskazuje miejsce, gdzie zapisana jest jakaś informacja (np. zmienna typu liczbowego czy struktura). Warto też przytoczyć w tym miejscu definicję adresu pamięci - możemy powiedzieć, że adres to pewna liczba całkowita, jednoznacznie definiująca położenie pewnego obiektu w pamięci komputera.*

Podstawy wskaźników

symbol	znaczenie	użycie
*	weź wartość x	*x
*	deklaracja wskaźnika do wartości	int *x;
&	weź adres	&x

a) Przykład operacji pobrania adresu:

```
double a=1; //definicja zmiennej typu double – rezerwacja 8 bajtów pamięci
double *b;  /*definicja wskaźnika do zmiennej typu double; rezerwacja 4 B;
              ten wskaźnik teraz nie jest ustawiony */
b = &a;     /*przypisanie zmiennej typu double * (wskaźnik do typu double)
              adresu zmiennej a; teraz zmienna b jest ustawiona*/
printf("rozmiar a = %d\n", sizeof(a)); // 8B
printf("rozmiar b = %d\n", sizeof(b)); // 4B dla systemów 32-bitowych
printf("a=%lf b=%p *b=%lf\n", a, b,*b);
```

b) Przykład operacji dostępu do danych po adresie (operator wskazania pośredniego):

```
double a = 3.14;
double *b = NULL; //b = 0x0000000
double c = 0.0; /*inicjalizacja c = 0.0*/
b = &a; /*b = 0x0064ff12*/
c = *b; /* c = 3.14; *b – to jest operator wskazania pośredniego*/
*b += a; /* *b – L-value; *b = *b+a; Pod adresem b teraz jest zmienna o
            wartości 6.28; */
c = *b; /* c = 6.28;*/
printf("a = %le\n", a); /* wydruk: a = 6.280000e+00;*/
```

Przykład umożliwia zmianę wartości zmiennej „a” poprzez operacje wykonane na jej adresie w pamięci.

Przekazywanie argumentów przez wskaźnik – polega na przekazaniu adresu (np. zmiennej), a nie samej wartości. Dzięki temu funkcja może wykonywać operacje na oryginalnym obiekcie.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
	int i;
	double y, al;
	double *z;
	double ff(double *);
	printf("Podaj liczbe rzeczywista ");
	scanf("%lf", &y);
	printf("Podaj liczbe calkowita ");
	scanf("%d", &i);
	z = &y;
	printf("z = %p\n", z);
	al = ff(&y); //lub al = ff(z);
	printf("Wyniki: %lf\n", al);
	printf("y=%lf\n", y);
	//  system("pause");
}
double ff(double *x)
{
	double gg;
	printf("x = %p\n", x);
	gg = 1 / (*x * *x + sqrt(1 + 2 * *x + 3 * *x * *x));
	*x = 12; //zmiana wartości y
	return gg;
}
```

6. **Przekazywanie argumentów do funkcji przez referencję**.
   Jest to niejawny sposób przekazywania zmiennych poprzez wskaźnik. Różni się formą zapisu. O referencji można myśleć jako o innej nazwie dla wskazywanego przez referencję obiektu.
   - przy definicji trzeba jednocześnie dokonać inicjalizacji, nie ma pustych referencji;
   - po inicjalizacji nie można już zmieniać „celu” referencji.

   Uwaga: w środowisku VS we właściwościach projektu ustawić opcje kompilacji: C/C++\Zaawansowane\Kompiluj jako: Domyślny

```
#include <stdio.h>

void fun_3(int &k);

int main()
{
	int i = 1;
	int &r = i;
	fun_3(i);
	printf("i=%d\n", i);
	r = 20;
	printf("i=%d\n", i);
}
void fun_3(int &k)
{
	k = 10 * k;
}
```

7. Sposoby przekazywania argumentów do funkcji – podsumowanie:

Poprzez wartość	Poprzez wskaźnik	Poprzez referencję
//prototyp void fun_1(int k);	*//prototyp* void fun_2(int *k);	*//prototyp* void fun_3(int &k);
//wywołanie poprzez wartość fun_1(i); *//wartość i **nie** zmieniła się*	*//wywołanie poprzez wskaźnik* fun_2(&i); *// wartość i zmieniła się*	*//wywołanie poprzez referencję* fun_3(i); *// wartość i zmieniła się*
//definicja funkcji void fun_1(int k) { k = 10*k; }	*//definicja funkcji* void fun_2(int *k) { *k = 10*(*k); }	*//definicja funkcji* void fun_3(int &k) { k = 10*k; }

8. Wskaźniki typ **void *** - definiowania wskaźników na dane dowolnego typu.
   *Nie ma zmiennych typu **void**, można jednak tworzyć wskaźniki typu **void ***. Ze względu na zgodność z językiem C, wskaźnikom takim można przypisać adres dowolnego obiektu (lub wartość dowolnego wskaźnika), i zrzutować na dowolny inny typ wskaźnika, obchodząc tym samym kontrolę typów wykonywaną przez kompilator.*

```
int n = 10;
void *p = &n;
int *pn = (int *)p;
printf("n = %d\n", *pn);
```

   Zdefiniowano wskaźnik **p* i przypisano mu adres zmiennej całkowitej *n*. Ponieważ kompilator nie ma żadnej informacji o typie danych wskazywanych przez *p* nie wolno bezpośrednio „wyłuskiwać" danych wskazanych przez *p*. Aby uzyskać dostęp do danych, należy jawnie wskazać na ich typ.

9. Napisz program, który dla podanego promienia, w funkcji ***void kolo (……)*** obliczy pole i obwód koła. Prototyp funkcji:
   `void kolo(double r, double *w1, double *w2);`

10. Napisz i przetestuj funkcję:
    `void zamien(int *x, int *y);`
    zamieniającą wartości dwóch zmiennych całkowitych (posługując się przekazywaniem przez zmienną).

11. Napisz funkcję, która policzy:

$$z = \begin{cases} x^2 + y \;\; dla\; x^2 + y^2 \leq 1 & (1) \\ 2x + y^2 \;\; dla\; y > x + 5 & (2) \\ \frac{1}{2}x^2 + y \;\; dla\; x^2 + y^2 > 1 \;\; i \;\; y \leq x + 5 & (3) \end{cases}$$

    oraz poda numer wzoru, wg którego wartość z została policzona.

12. Napisz program, który przy pomocy rekurencji wyznaczy sumę n kolejnych liczb naturalnych (liczba n podawana przez użytkownika; suma obliczana w dedykowanej funkcji). Korzystając z dokumentacji MSDN zapoznaj się z pojęciem rekurencji:
    https://docs.microsoft.com/en-us/cpp/c-language/recursive-functions?view=vs-2017

13. Znaleźć rozwiązanie problemu Wież z Hanoi. Ta starożytna łamigłówka pochodzi z pewnego klasztoru w Tybecie:

    Przypuśćmy, że mamy trzy wieże lub, skromniej, trzy kołki, A, B i C. Na pierwszym kołku A, znajdują się trzy krążki nanizane w porządku malejących wielkości, podczas gdy pozostałe kołki są puste. Chcemy przenieść krążki z kołka A na B, być może używając do tego kołka C. Według reguł gry krążki można przenosić po jednym na raz i w żadnej chwili krążek większy nie może być umieszczony na wierzchu mniejszego.
    Wskazówka: Przedstawiony tu algorytm radzi sobie z przeniesieniem N krążków z kołka A przez C na kołek B następująco. Najpierw sprawdza, czy N=1, w którym to przypadku po prostu przenosi na miejsce przeznaczenia ten jedyny krążek, z którym miał się uporać (lub, dokładniej, podaje opis jednego ruchu, który to zrobi) i natychmiast powraca. Jeśli N>1, algorytm najpierw przenosi górne N-1 krążków z kołka A na kołek "pomocniczy" C, używając rekurencyjnie tego samego podprogramu; potem bierze jedyny krążek pozostawiony na kołku A (musi to być krążek największy - dlaczego?) i przenosi go na ostateczne miejsce przeznaczenia, na kołek B; potem, znowu rekurencyjnie, przenosi N-1 krążków, które poprzednio "przechował" na kołku C, na ich ostateczne miejsce, na kołek B. Napisać rekurencyjną funkcję:

    ```
    void hanoi(int N, char A, char B, char C);
    ```

    realizującą ten algorytm i wydrukować wynik dla trzech krążków.

14. Wartości domyślne argumentów funkcji.
    Podczas deklaracji funkcji można podać wartości domyślne argumentom funkcji. Jeżeli deklaracja jest połączona z definicją funkcji, wartości domyśle argumentom podajemy podczas definicji. Wymagania:
    - Wartości domyślne przypisujemy argumentom na końcu listy,
    - Jeżeli w deklaracji funkcji podałeś wartość domyślną dowolnemu argumentowi funkcji to wartości domyślne należy nadać wszystkim kolejnym argumentom,
    - Jeżeli przy wywołaniu funkcji pominiemy argument aktualny, któremu przypisano wartość domyślną, to należy pominąć wszystkie kolejne argumenty aktualne z listy argumentów (musimy pozostawić wartości domyślne tych argumentów)
    - **Uwaga: w środowisku VS we właściwościach projektu ustawić opcje kompilacji: C/C++\Zaawansowane\Kompiluj jako: Domyślny**

    Przykład.

    ```
    #include <stdio.h>
    void print(int tab[], int rozmiar, char znak = ',', char znak_k = ';') {
        int i;
        for (i = 0; i < rozmiar - 1; i++)
            printf("%d%c", tab[i], znak);
        printf("\b%c\n", znak_k);
    }

    int main() {
        int tab[] = { 0,1,2,3,4,5,6,7,8,9 };

        print(tab, 10);             //dwie wartości domyślne
        print(tab, 10, '_');        //jedna wartość domyślna
        print(tab, 10, '_', ':');   //nie korzystamy w wartości domyślnych

        return 0;
    }
    ```

15. Funkcje o dowolnej/zmiennej liczbie argumentów.
Funkcje o nieokreślonej liczbie argumentów deklarujemy używając wielokropka (przecinek z trzema kropkami) (, ...) w miejsce parametrów, których liczby nie specyfikujemy. Kontrola typów nie jest wykonywana na takich argumentach. Co najmniej jeden parametr musi poprzedzać notację wielokropka, a notacja wielokropka musi stanowić ostatni element na liście parametrów formalnych. Aby korzystać z tego mechanizmu należy dołączyć plik nagłówkowy <stdarg.h>. Przed wielokropkiem wymienione są stałe (obowiązkowe) argumenty funkcji.

Przykład 1:

```
#include <stdio.h>
#include<stdarg.h>
int max(int ile, int jeden, ...)
{
    va_list arg;
    va_start(arg, jeden);
    int max = jeden;
    for (int i = 2; i <= ile; i++) {
        int a = va_arg(arg, int);
        if (a > max) max = a;
    }
    va_end(arg);
    return max;
}
int main() {
    printf("%d\n", max(9, 1, 2, 3, 4, 5, 60, 7, 8, 9));
    return 0;
}
```

W pliku nagłówkowym stdarg.h zdefiniowany jest typ `va_list` (variable-arguments list) oraz trzy związane z nim funkcje `va_start()`, `va_arg()` i `va_end()`.
Tok postępowania:

- Tworzymy zmienną typu `va_list arg;`
- Wywołujemy funkcję `va_start(arg, jeden)` podając jako pierwszy argument utworzoną wcześniej zmienną typu `va_list` oraz drugi argument będący nazwą zmiennej odpowiadającej ostatniemu argumentowi stałemu (obowiązkowemu) w definiowanej funkcji.
- Wartości kolejnych argumentów odczytujemy poprzez wywołanie funkcji `va_arg(arg,int)` podając jako pierwszy argument utworzoną wcześniej zmienną typu `va_list` oraz drugi argument będący typem wartości na jaki odczytane dane mają być przekształcone (typ odczytywanej wartości musi być znany).
- Wywołujemy funkcję `va_end(arg)` podając jako argument utworzoną wcześniej zmienną typu `va_list` (porządkowanie stosu, wywołanie tej funkcji jest konieczne!).

Przykład 2 – jak podać typy wartości przekazywanych do funkcji z dowolną liczbą argumentów:

```c
#include <stdio.h>
#include <stdarg.h>
void typ_arg( const char typ[], ...);

int main() {
    typ_arg("iSdDIs", 16, "Nowak", 2.5, 14.3, 3, "Piotr");
    typ_arg("iDL", 15, 9.5, 12);
}

void typ_arg(const char typ[], ...) {
    int typ_int, i = 0;
    char* typ_string, c;
    double typ_double;

    va_list arg;
    va_start(arg, typ);

    while ((c = typ[i++]) != '\0') {
        switch (c) {
        case 'i':
        case 'I':
            typ_int = va_arg(arg, int);
            printf("Typ int: %d\n", typ_int);
            break;
        case 'd':
        case 'D':
            typ_double = va_arg(arg, double);
            printf("Typ double: %lf\n", typ_double);
            break;
        case 's':
        case 'S':
            typ_string = va_arg(arg, char*);
            printf("Typ char*: %s\n", typ_string);
            break;
        default:
            printf("Nieznany kod typu! \n");
            goto KONIEC;
        }
    }
    KONIEC:
    printf("\n");
    va_end(arg);
}
```

16. Przeanalizuj w trybie pracy krokowej następujące przykłady do wykładów:
    https://torus.uck.pk.edu.pl/~fialko.sergiy /text/CC/przykl/WW3_1.pdf
    https://torus.uck.pk.edu.pl/~fialko.sergiy /text/CC/przykl/WW3_2.pdf

**Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.*