Laboratorium 3: „Funkcje”

dr inż. Leszek Ciopiński
dr inż. Arkadiusz Chrobot
dr inż. Grzegorz Łukawski

4 września 2026

1. Wprowadzenie

Pierwsza część instrukcji zawiera informacje o sposobie tworzenia i korzystania z funkcji w języku C. Druga część traktuje o zmiennych lokalnych, trzecia zawiera informacje na temat posługiwania się parametrami funkcji. W części czwartej opisano zagadnienia związane z funkcjami rekurencyjnymi.

2. Funkcje

Język C jest językiem wspierającym strukturalny model programowania. Realizacją koncepcji podprogramu w tym języku jest funkcja.

2.1. Prototyp i definicja funkcji

Definicja funkcji w języku C składa się z nagłówka i treści nazywanej też *ciałem funkcji*. Nagłówek rozpoczyna się deklaracją typu wartości zwracanej przez funkcję. Jeśli funkcja nic nie zwraca, a więc zachowuje się podobnie do procedury z języka Pascal, to w tym miejscu deklaracji należy użyć typu `void`. Kolejnym elementem nagłówka jest nazwa funkcji. Po nazwie, w nawiasach okrągłych umieszczana jest lista parametrów funkcji, która składa się z ich deklaracji rozdzielonych przecinkami. Jeśli funkcja nie będzie przyjmowała żadnych argumentów wywołania, to jej lista parametrów powinna być pusta, co zaznaczamy umieszczając w nawiasach słowo kluczowe `void`. Innym sposobem na poinformowanie kompilatora, że funkcja nie ma parametrów jest pozostawienie wnętrza nawiasów pustego. Ten sposób nie oznacza jednakże, że lista argumentów musi pozostać pusta, ale że funkcja przyjmuje nieokreśloną liczbę argumentów wywołania. Treść funkcji, a więc deklaracje zmiennych lokalnych, definicje innych elementów lokalnych i instrukcje do wykonania ujęte są w parę nawiasów klamrowych. W języku C funkcje można również deklarować. Deklaracja funkcji polega na zdefiniowaniu jej prototypu, czyli nagłówka zakończonego średnikiem. Dla tak zadeklarowanej funkcji należy napisać definicję, jednak nie musi ona być umieszczona tuż po prototypie. Może być nawet umiejscowiona za funkcją `main()`. Dodatkowo w prototypie można nie umieszczać nazw parametrów funkcji, a jedynie ich typy. Język C nie pozwala na zagnieżdżanie definicji funkcji.

2.2. Zwracanie wartości

Jeśli typ wartości zwracanej przez funkcję jest różny od `void`, to w ciele funkcji musi wystąpić słowo kluczowe `return`, po którym znajdzie się wartość mieszcząca się w tym typie. Funkcja zwróci właśnie tę wartość. Można ją zapisać w nawiasach okrągłych, ale nie jest to wymagane. Zamiast wartości, po słowie kluczowym `return` może wystąpić także stała, zmienna lub całe wyrażenie. Oprócz zwrócenia wartości słowo to powoduje także zakończenie wykonania funkcji, dlatego ważnym jest, aby każdy możliwy sposób wykonania funkcji kończył się tym słowem kluczowym. Inaczej funkcja może zwrócić nieokreśloną wartość. W funkcjach, które deklarują typ wartości zwracanej jako `void` może również wystąpić słowo kluczowe `return`, jednakże po nim powinien wystąpić jedynie średnik. W takim przypadku działanie tego słowa ogranicza się do zakończenia wykonania funkcji.

2.3. Wywołanie funkcji

Jeżeli w definicji funkcji podano listę parametrów, to w miejscu jej wywołania muszą pod te parametry być podstawione argumenty wywołania. Jeżeli funkcja nie przyjmuje żadnych parametrów, to po jej nazwie w miejscu wywołania musi wystąpić pusta para okrągłych nawiasów, które nazywane są *operatorem wywołania funkcji*. Jeśli funkcja zwraca wartość, to może ona zastać przypisana w miejscu wywołania funkcji do zmiennej o kompatybilnym typie. Można jej również nigdzie nie przypisywać, wtedy mamy do czynienia z tak zwanym *wywołaniem funkcji ze względu na efekt uboczny jej wykonania*. W takim przypadku niektórzy programiści rzutują wartość wywoływanej funkcji na typ `void`. Nie jest to jednak konieczne. Wywołanie w wyrażeniu funkcji, która nic nie zwraca lub próba przypisania do zmiennej jej wartości jest błędem. Listing 1 pokazuje kilka sposobów tworzenia i wywołania prostych funkcji bez parametrów. Funkcje z parametrami będą opisane w dalszej części instrukcji.

```c
#include<stdio.h>

int f1(void)
{
        puts("Jestem funkcją f1(), zwracam wartość 5.");
        return(5);
}

int f2()
{
        puts("Jestem funkcją f2(), zwracam wartość 4.");
        return 4;
}

void f3(void)
{
        puts("Jestem funkcją f3() i nic nie zwracam.");
}

void f4()
{
        puts("Jestem funkcją f4() i też nic nie zwracam.");
}

void f5(void)
{
        puts("Jestem funkcją f5().");
        return;
        puts("Nic nie zwracam i nie wypiszę tego komunikatu.");
}

void f6(void); //Prototyp funkcji

int main(void)
{
        int a=f1(); //Wywołujemy funkcję f1() i przypisujemy wartość, którą
                    //zwróciła do zmiennej a.
        printf("Wartość zmiennej a: %d\n",a);
        (void)f1(); //Wywołujemy funkcję f1() i ignorujemy to co zwróciła.
        f2();       //Wywołujemy funkcję f2() i ignorujemy to co zwróciła.
        //a=f3();   Tak nie wolno wywołać funkcji f3()!
        f3();
        f4();
        f5();
        f6();
        return 0;
}

void f6(void)
{
        puts("Jestem funkcją f6(). Zostałam zdefiniowana po funkcji main().");
}
```

Listing 1: Funkcje bez parametrów

3. Zmienne lokalne

Zmienne lokalne w języku C nazywane są również zmiennymi automatycznymi. Mogą być deklarowane praktycznie w każdym miejscu funkcji. Należy pamiętać, że nie są one domyślnie inicjowane żadną ustaloną wartością i to programista odpowiedzialny jest za nadanie im określonej wartości początkowej. Zmienne lokalne tworzone są na stosie, ale dzięki słowu kluczowemu `register` możemy nakazać kompilatorowi, aby umieścił ją w jednym z rejestrów procesora. Ma to na celu przyspieszenie wykonania programu. Opisywane zmienne są niszczone po zakończeniu funkcji. Jeśli chcielibyśmy, aby między kolejnymi wywołaniami funkcji te zmienne istniały, bo np. jest potrzebna nam wartość w nich umieszczona, to możemy w ich deklaracji użyć słowa kluczowego `static`. Zmienna tak oznaczona jest praktycznie zmienną globalną (jej wartość początkowa wynosi zawsze zero), ale widoczną tylko wewnątrz funkcji. Listing 2 zawiera przykłady deklaracji i użycia zmiennych lokalnych.

```c
#include<stdio.h>

//Kompiluje się z ostrzeżeniami!
void f1(void)
{
        int a;
        printf("Ciekawe jaką wartość ma a: %d\n",a);
        register int b;
        printf("A teraz kolej na b: %d\n",b);
        static int c;
        printf("Ta funkcja była wcześniej wywoływana %d razy.\n",c);
        c++;
        a++;
        b=5;
}

int main(void)
{
        int i;
        for(i=0;i<3;i++)
                f1();
        return 0;
}
```

Listing 2: Zmienne lokalne

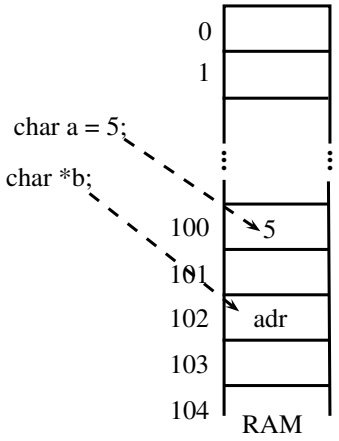
4. Wskaźniki

Ponieważ programowanie komputerów często związane jest z ich architekturą, do zrozumienia dalszej części instrukcji konieczna jest krótka dygresja o organizacji pamięci RAM i jej adresowaniu.

Jak wiadomo, zmienne używane w języku programowania fizycznie są umieszczane w pamięci operacyjnej komputera. Pamięć RAM, pomimo że może przechowywać różne typy danych, ma prostą budowę, która pomimo pewnych zmian, wywodzi się jeszcze z czasów systemów 8-bitowych. Dlatego z punktu widzenia programisty, pamięć RAM jest długą listą składającą się z komórek pamięci. Każda komórka ma swój unikalny adres i może przechowywać jeden bajt danych, czyli 8 bitów. W związku z tym zmienna typu *char* zajmuje tylko jedną komórkę, podczas gdy zmienna typu *long int* zajmuje 8 kolejnych komórek, a więc i 8 adresów w pamięci. Na etapie kompilacji zmienna zastępowana jest adresem do pamięci, gdzie ma znajdować się dana wartość. Proces ten jest bardziej złożony, jednak jego szczegóły nie będą potrzebne do zrozumienia dalszej części instrukcji.

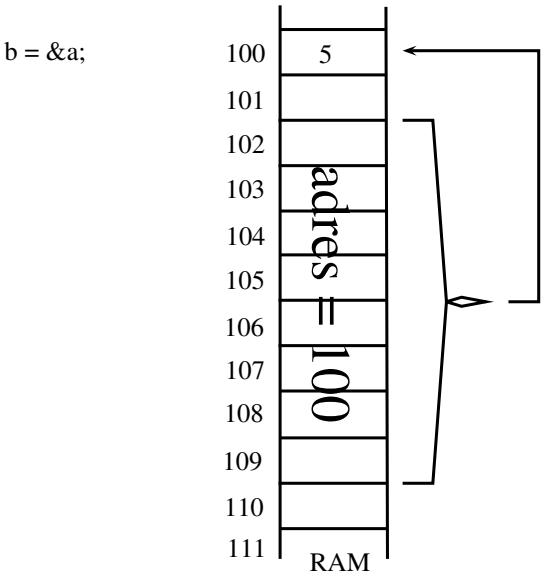
Wskaźnik jest zmienną, która przechowuje adres do pamięci. Dla systemów 32-bitowych jej rozmiar

wynosi 4 bajty, a dla systemów 64-bitowych już 8 bajtów. Wskaźnik może wskazywać na zmienną dowolnego typu, jednak jest to bez wpływu na jego rozmiar. Do deklaracji typu wskaźnikowego używamy symbolu asterysk „`*`”, popularnie określanym jako „gwiazdka”.



Rysunek 1: Odwzorowanie zmiennych w pamięci. Numeracja pamięci użyta jest wyłącznie do zwiększenia przejrzystości opisu i może ulec zmianie przy każdym uruchomieniu programu.

Przykład odwzorowania zmiennej i wskaźnika przedstawiony został na Rysunku 1. Zmienna `a` umieszczona została w pamięci RAM pod adresem 100. Zakładając, że użyty został system 64-bitowy, w komórkach o adresach od 102 do 109 włącznie zarezerwowane jest miejsce na adres. Ponieważ podczas deklarowania wskaźnika nie została zadeklarowana jego wartość, zawartość tych komórek jest przypadkowa (jeżeli wskaźnik zadeklarowano jako zmienną lokalną) lub ustawiona na 0 (jeżeli wskaźnik zadeklarowano jako zmienną globalną). Do pobierania adresu innej zmiennej służy operator ampersand `&`, którego użycie przedstawiono na Rysunku 2.



Rysunek 2: Wyłuskanie adresu danej zmiennej. W wyniku tej operacji liczba zapisana w komórkach od 102 do 109 wynosi 100.

Listing 3 przedstawia praktyczne działanie wskaźników. Instrukcja `b=&a;` powoduje przypisanie do zmiennej `b` adresu do pamięci, pod którym znajduje się zmienna `a`. Dlatego drugie wywołanie funkcji `printf()` nie wyświetla wartości 5, a adres. Ponieważ aktualny adres zmiennej określany jest przez system operacyjny, przy każdym uruchomieniu programu może on mieć inną wartość. Aby pobrać wartość, na którą wskazuje wskaźnik, konieczne jest użycie operatora asterysk (*). Przykład jego użycia zaprezentowano w trzecim wywołaniu funkcji `printf()`. W związku z tym, że wskaźnik nie jest kopią

innej zmiennej, ale jest adresem do niej, zmiana wartości zmiennej `a` skutkuje wyświetleniem wartości 7 w ostatnim wywołaniu funkcji `printf()`.

```c
#include<stdio.h>

int main(void)
{
        char a = 5;
        char *b;

        b=&a;
        printf("%d\n", a);   //Wyświetla 5
        printf("%ul\n", b);  //Wyświetla aktualny adres
        printf("%d\n", *b);  //Wyświetla 5

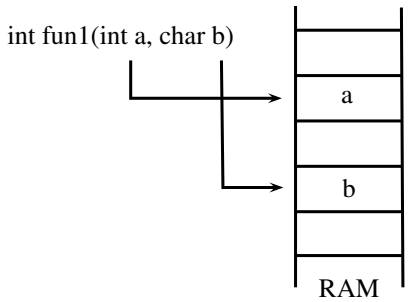
        a=7;
        printf("%d\n", *b);  //Wyświetla 7

        return 0;
}
```

Listing 3: Przykład działania wskaźników.

5. Parametry

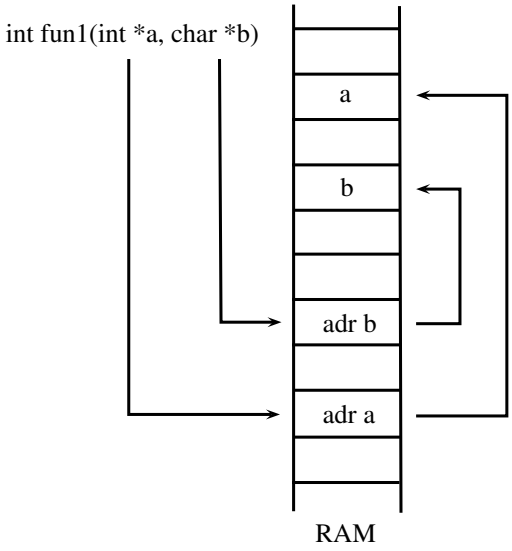
Parametry są też zmiennymi lokalnymi, ale takimi, które dostępne są na zewnątrz funkcji i służą do wymiany informacji z jej otoczeniem. Nie mogą one mieć zatem takich samych nazw jak zmienne lokalne, ale mogą mieć te same identyfikatory co zmienne globalne. W języku C podstawowym sposobem przekazania informacji do funkcji przez parametr jest przekazanie przez wartość. Jeśli chcemy, aby parametr był tylko do odczytu, to poprzedzamy jego deklarację słowem kluczowym `const`. W przypadku braku tego słowa kluczowego, na etapie wywołania funkcji tworzone są kopie wartości przekazywanych jako parametry, co prezentuje Rysunek 3. Dlatego wartości te mogą być zmieniane wewnątrz funkcji, jednak zmiana ta nie będzie widoczna po jej zakończeniu.



Rysunek 3: Odzwierciedlenie parametrów funkcji w pamięci.

Jeśli chcemy za parametr podstawić zmienną, której wartość w trakcie wykonania funkcji ma być zmodyfikowana i utrwalona po jej zakończeniu, to parametr musimy zadeklarować jako wskaźnik (po nazwie typu, a tuż przed nazwą parametru musi wystąpić znak `*`). Oznacza to, że jako parametr do funkcji przekazane zostaną wskaźniki na zmienne. Podobnie jak w poprzednim przypadku, również ich wartości zostaną skopiowane. Ponieważ wskazywać one będą na zmienne zadeklarowane poza funkcją, użycie

wskaźników do ich modyfikacji spowoduje, że zmiany te pozostaną również po zakończeniu działania funkcji. Sytuacje tą prezentuje Rysunek 4.



Rysunek 4: Zmiana wartości zmiennych `a` lub `b` poprzez wskaźniki pozostanie widoczna również po zakończeniu działania funkcji.

W ciele funkcji odwołujemy się do wartości zmiennej wskazywanej przez ten wskaźnik poprzedzając nazwę wskaźnika znakiem `*`. W miejscu wywołania funkcji, za taki parametr podstawiamy zmienną, której nazwę poprzedzamy znakiem `&`, który w tym wypadku oznacza tak zwany *operator wyłuskania*, czyli operator zwracający adres zmiennej. Listing 4 ilustruje kilka sposobów użycia parametrów funkcji.

```
#include<stdio.h>

int f1(int a, int b)
{
        return a+b;
}

void f2(int a)
{
        printf("Funkcja f2() dostała wartość: %d",a);
        a=10;
        printf(" i zmieniła ją na: %d.\n",a);
}

void f3(int *a)
{
        printf("Funkcja f3() dostała wartość: %d",*a);
        *a=10;
        printf(" i zmieniła ją na: %d.\n",*a);
}

void f4(const int a)
{
        printf("Funkcja f4() dostała wartość %d, której nie może zmienić.\n",a);
}

int main(void)
```

```
{
        //Funkcję zwracającą wartość możemy wywołać wewnątrz wyrażenia.
        printf("Wynik funkcji f1(): %d\n",f1(5,3));
        int a=3;
        printf("Wartość zmiennej a przed wywołaniem funkcji f2(): %d\n",a);
        f2(a);
        printf("Wartość zmiennej a po wywołaniu funkcji f2(): %d\n",a);
        printf("Wartość zmiennej a przed wywołaniem funkcji f3(): %d\n",a);
        f3(&a);
        printf("Wartość zmiennej a po wywołaniu funkcji f3(): %d\n",a);
        printf("Wartość zmiennej a przed wywołaniem funkcji f4(): %d\n",a);
        f4(a);
        return 0;
}
```

Listing 4: Parametry

6. Zadania

KAŻDY PROGRAM NALEŻY NAPISAĆ Z PODZIAŁEM NA FUNKCJE Z PARAMETRAMI! Część zadań jest powtórzona z wcześniejszych instrukcji, ale teraz należy je zrealizować z podziałem na funkcje.

1. Użyj debuggera, aby wykonać krokowo programu z Listingu 1.
2. Użyj debuggera, aby wykonać krokowo program z Listingu 4 i prześledzić jak zmieniają się wartości parametrów i zmiennych.
3. Napisz program, który będzie znajdował największą spośród trzech podanych przez użytkownika liczby całkowitych.
4. Napisz funkcję wyświetlającą na ekranie nazwę dnia tygodnia, którego numer podano przez parametr.
5. Napisz program, który w zależności od wyboru użytkownika, będzie liczył sumę ciągu arytmetycznego lub geometrycznego.
6. Napisz program, który policzy pole powierzchni prostopadłościanu o podanej przez użytkownika wysokości, szerokości i głębi.
7. Napisz program, który policzy pole powierzchni ostrosłupa foremnego, którego podstawą jest kwadrat. Wymiary, czyli długość boku podstawy, oraz wysokość ostrosłupa podaje użytkownik.
8. Napisz program, który policzy z zadaną przez użytkownika dokładnością wartość liczby PI.
9. Napisz program, który policzy z zadaną dokładnością wartość pierwiastka kwadratowego niezerowej liczby naturalnej. Wartość dokładności i liczby powinny być podane przez użytkownika.
10. Napisz program z funkcją, która będzie liczyła resztę z dzielenia dwóch niezerowych liczb naturalnych. Funkcja może używać do tego działania jedynie operatora odejmowania.
11. Napisz program, który pobierze od użytkownika ciąg składający się z liter „a” i „b”, który ma długość 10 znaków (każdy znak musi być wczytany z osobna). Program powinien wykryć i zasygnalizować, jeśli w tym ciągu znajdzie się sekwencja „abba”.