

Wykład 2: Typy danych. Typy zmiennych.

dr inż. Andrzej Stafiniak

Wrocław 2024



HR EXCELLENCE IN RESEARCH



Politechnika Wroclawska

Struktura programu w języku C – postać źródłowa

Dyrektywy preprocesora

```
#include <stdio.h>
#include "biblioteka.h"
#include "file.c"
#define PI 3.14
#define Pow(x) ((x)*(x))
```

```
// standardowa biblioteka C -
// - operacje we-wy
```

Komentarze

```
//...., /* ...*/
```

```
// makro x^2
```

Deklaracje funkcji i zmiennych

```
float funkcja (int, int);
```

```
//prototyp funkcji
```

Funkcja główna main()

```
int main()
{
    instrukcje;
    instrukcje;
    return 0;
}
```

```
// int main(int argc, char *argv[]),
// int main(void), main()
```

Definicja innych funkcji

```
float funkcja (int x, int y){
    return x + y;
}
```

main():

- występuje w każdej konsolowej aplikacji,
- jest zawsze pierwszą wywołaną funkcją,
- wszystko co dzieje się w programie jest określone lub wywołane w ciele tej funkcji

Struktura programu w języku C – postać źródłowa

Nagłówek funkcji

Sygnatura funkcji

Nazwa funkcji

Lista argumentów

Typ zwracany

Funkcja główna
main()

```
int main()
{
    instrukcje;
    instrukcje;
    return EXIT_SUCCESS;
}
```

Ciało funkcji

```
float funkcja (int x, int y) {
    return x + y;
}
```

Typy instrukcji:

- deklaracje;
- przypisanie;
- wywołanie funkcji;
- instrukcje sterujące;
- instrukcja pusta;

Czym jest wartość zwracana
prze funkcję `int main()` ?

Identyfikatory / nazwy

- Służą one do nazwania każdego obiektów/elementów programu, jak: typy, stałe, zmienne, instrukcje, funkcje itp. ...
- Są ciągiem liter, cyfr i znaków „_”. Identyfikator musi rozpoczynać się od litery albo od symbolu podkreślenia.
- Nie może zawierać odstępu lub znaku specjalnego.
- Nie może rozpoczynać się od dwóch znaków podkreślenia (możliwość kolizji z nazwami używanymi w kompilatorach).
- Wybierając identyfikator staramy się o nazwę mającą znaczenie mnemotechniczne, np. *voltageValue*
- Rozróżniane są duże i małe litery

P. Mikołajczak, Język C – podstawy programowania, UMCS, Lublin 2011

Konwencje nazewnictwa zmiennych i funkcji

camelCase

np. *floatingPointValue*

snake_case

np. *floating_point_value*

Identyfikatory / nazwy

- Niektóre nazwy zostały zastrzeżone przez twórców języka.
- Nazywają się **słowami kluczowymi**.
- Nie mogą być używane jako nazwy zmiennych lub funkcji, ponieważ mają specjalne znaczenie.

Słowa kluczowe języka C (standard ANSI C oraz standard C9X)

auto	double	inline	static
break	else	int	struct
case	enum	long	switch
char	extern	register	typedef
complex	float	restrict	union
const	for	return	unsigned
continue	goto	short	void
default	if	signed	volatile
do	imaginary	sizeof	while

Dobre nawyki

- Spójny zestaw reguł dotyczących formatowania kodu w celu poprawy jego czytelności:
 - wcięcia w kodzie;
 - stosowanej konwencji notacji;
 - organizacja kodu w bloki;
 - ograniczona długość linii kodu;
 - dokumentacji kodu w postaci komentarzy.

Michał Stępiak

Dobre nawyki

➤ Spójny zestaw reguł dotyczących formatowania kodu w celu poprawy jego czytelności:

- wcięcia w kodzie;
- stosowanej konwencji notacji;
- organizacja kodu w bloki;
- ograniczona długość linii kodu;
- dokumentacji kodu w postaci komentarzy.

```
int main(){ costam; drugiecostam;
abc(x,
y); return 0;}
int abc (int a, int b){
if(b!=0)
return a/b;
else{printf("Incorrect
data"); exit(EXIT_FAILURE);}}
```

```
int main()
{
    deklaracje; // int x;
    instrukcje; // scanf();
    div(x,y);
    return 0;
}
```

```
/*Realizacja operacji dzielenia*/

int div (int a, int b){
    if(b!=0)
        return a/b;
    else{
        printf("Error");
        exit(EXIT_FAILURE);
    }
}
```

Michał Stępnia

Dobre nawyki

➤ Spójny zestaw reguł dotyczących formatowania kodu w celu poprawy jego czytelności:

- wcięcia w kodzie;
- stosowanej konwencji notacji;
- organizacja kodu w bloki;
- ograniczona długość linii kodu;
- dokumentacji kodu w postaci komentarzy.

```
int main(){ costam; drugiecostam;
abc(x,
y); return 0;}
int abc (int a, int b){
if(b!=0)
return a/b;
else{printf("Incorrect
data"); exit(EXIT_FAILURE);}}
```

ch... widać! ;)

```
int main()
{
    deklaracje; // int x;
    instrukcje; // scanf();
    div(x,y);
    return 0;
}
```

```
/*Realizacja operacji dzielenia*/

int div (int a, int b){
    if(b!=0)
        return a/b;
    else{
        printf("Error");
        exit(EXIT_FAILURE);
    }
}
```

Michał Stępnia

Dobre nawyki

- Spójny zestaw reguł dotyczących formatowania kodu w celu poprawy jego czytelności:
 - wcięcia w kodzie;
 - stosowanej konwencji notacji;
 - organizacja kodu w bloki;
 - ograniczona długość linii kodu;
 - dokumentacji kodu w postaci komentarzy.
- Dobrym zwyczajem jest deklarowanie wartości liczbowych jako stałych opatrzonych jednoznaczną nazwą.
- We wczesnych fazach rozwoju oprogramowania należy się skupić na wytworzeniu czytelnego, zrozumiałego, łatwego w utrzymaniu i w miarę możliwości zwięzłego kodu.
- Unikaj powtórzeń, dotyczy to zarówno duplikowania kodu oraz wykonywania powtarzających się czynności. Wydziel powtarzające się fragmenty kodu do osobnych funkcji, makr lub szablonów
- Funkcja powinna mieć tylko jedną odpowiedzialność. Kod należy dzielić na jednostki translacji ze względu na funkcjonalność.

Michał Stępiak

Typy danych - stałe

Stałe (ang. constraints) – to obiekty programu, których wartość w trakcie wykonywania programu nie może ulec zmianie. Nadaje się jej wartość w trakcie tworzenia programu. Typ stałej całkowitej czy rzeczywistej zależy od jej postaci, wartości i przyrostka.

- Dla liczby całkowitej domyślnie – `int`, `long int` lub `unsigned long int`. Możemy wymusić zmianę reprezentowania stałej całkowitej odpowiednim specyfikatorem `u/U`, `l/L`, `ll/LL` np. `17ul`, `555555L`.
- Dla liczby rzeczywistej domyślnym typem jest `double`, a specyfikatory `f/F`, `l/L` wymuszają typ `float` albo `long double` np. `155.1f` lub `1.1111112L`

I co istotne, w zależności od typu, dane zapisywane będą na różnej ilości bajtów, zależy to od konkretnego kompilatora.

```
sizeof(10) != sizeof(10ll)
```

Typy danych - stałe

Mamy do dyspozycji dwie metody tworzenia stałych:

- za pomocą dyrektywy `#define` – np. `#define NEWCONST 12`
- za pomocą słowa kluczowego `const` – np. `const float PI = 3.1415;`

`#define` jest dyrektywą preprocesora, który tworzy stałą jeszcze przed kompilacją i zamienia każdy ciąg tekstu `NEWCONST` na liczbę 12. Kompilator nie widzi już w kodzie ciągu `NEWCONST` tylko samą liczbę.

Warto także zauważyć, że nie podajemy typu stałej w przeciwieństwie do drugiej metody z wykorzystaniem słowa kluczowego `const`.

Typy danych - zmienne

Zmienne (ang. variables) – reprezentują dane identyfikowane nazwą i przechowywane są w obszarze pamięci operacyjnej, która może być wykorzystywana przez oprogramowanie. Zmienne muszą posiadać/charakteryzują się:

- nazwą (identyfikator),
- określony typ, który deklaruje rozmiar i układ pamięci zmiennej,
- zakres wartości, które mogą być przechowywane w obszarze pamięci zmiennej (określa typ),
- zestaw operacji, które można zastosować do zmiennej.

Zmienne przed użyciem muszą być **zadeklarowane!**

Zmienne – deklaracja

Deklaracja – jest to zapowiedź zmiennej, podająca informację dla kompilatora że jeżeli napotka on ciąg znaków, będący nazwą zmiennej, to ta zmienna będzie danego typu. Informacje te są potrzebne do rezerwacji odpowiedniej ilości miejsca w pamięci

```
typZmiennej nazwaZmiennej;
```

```
int x;
```

```
char litera;
```

```
float wartość;
```

- Tradycyjnie w starszych standardach języka C wymagano aby deklaracje wykonywać na początku bloku. Standard C99 i wyżej zezwala na umieszczenie deklaracji w dowolnym miejscu.
- W przypadku programowania z wykorzystaniem w wiele plików - często występuje sytuacja, że dana zmienna/funkcja jest zdefiniowana w jednym pliku, a następnie musi być używana przez inny plik, wtedy korzystamy z plików nagłówkowych z deklaracjami.

Zmienne – deklaracja

Deklaracja – jest to zapowiedź zmiennej, podająca informację dla kompilatora że jeżeli napotka on ciąg znaków, będący nazwą zmiennej, to ta zmienna będzie danego typu. Informacje te są potrzebne do rezerwacji odpowiedniej ilości miejsca w pamięci

```
typZmiennej nazwaZmiennej;
```

```
int x, y;
```

```
char litera;
```

```
float wartość;
```

Możliwość wielokrotnej deklaracji

- Tradycyjnie w starszych standardach języka C wymagano aby deklaracje wykonywać na początku bloku. Standard C99 i wyżej zezwala na umieszczenie deklaracji w dowolnym miejscu.
- W przypadku programowania z wykorzystaniem w wiele plików - często występuje sytuacja, że dana zmienna/funkcja jest zdefiniowana w jednym pliku, a następnie musi być używana przez inny plik, wtedy korzystamy z plików nagłówkowych z deklaracjami.

Zmienne – definicja


Definicja – definicja zmiennej jest to deklaracja zmiennej wraz z inicjalizacją, czyli wraz z procesem nadania wartości i rezerwacją miejsca w pamięci komputer. Odwołanie się do zmiennych lokalnych zadeklarowanych, ale niezainicjalizowanych, da nieokreśloną wartość (losową).

```
typZmiennej nazwaZmiennej = wartość;
```

```
int x = 77;
```

```
char litera = 'c';
```

```
float wartośćZmiennoprzecinkowa = 1.333;
```



Operator przypisania

Typy danych

Typy danych:

- **proste** (fundamentalny, wbudowany)
- **pochodne** (tworzone na podstawie typów podstawowych i operatorów jak *, [], ())

Elementarne typy **proste**:

- **char** – pojedynczy znak, mała liczba całkowita
- **int** – liczby całkowite,
- **float** – liczby rzeczywiste (zmiennoprzecinkowe)
- **double** – liczby rzeczywiste o podwójnej precyzji.

Modyfikatory typów, niektóre będące również typem:

- **short** – liczba krótka,
- **long** – liczba długa,
- **signed** – liczba ze znakiem,
- **unsigned** – liczba bez znaku.

←
słowa kluczowe

Typy proste

Słowo kluczowe	Rozmiar [B]	Zakres wartości	Zastosowanie
<code>char</code>	1	[-128; +127]	Tablica ASCII, małe liczby
<code>unsigned char</code>	1	[0; 255]	Rozszerzone ASCII, małe liczby
<code>short</code>	min. 2	min. [-32768; +32767]	małe liczby, pętle
<code>unsigned short</code>	min. 2	min. [0; 65535]	małe liczby, pętle
<code>int</code>	min. 2	min. [-32768; +32767]	liczby, pętle
<code>unsigned int</code>	min. 2	min. [0; 65535]	liczby, pętle
<code>long</code>	min. 4	min. [± 2147483648]	duże liczba
<code>unsigned long</code>	min. 4	min. [0; 4294967295]	duże liczba
<code>long long</code>	min. 8	[± 9223372036854775808]	duże liczba
<code>unsigned long long</code>	min. 8	[0; 18446744073709551615]	duże liczba
<code>float</code>	typ. 4	typ. $3.4E \pm 38$	obl. naukowe, dokł. do 7 znaków po przecinku
<code>double</code>	typ. 8	typ. $1.7E \pm 308$	obl. naukowe, dokł. do 15 znaków po przecinku
<code>long double</code>	typ. 12	typ. $1.1E \pm 4932$	obliczenia finansowe i naukowe
<code>bool*</code>	1	true/false	instrukcje sterujące
<code>void</code>	0	brak wartości	

* stdbool.h



Typy całkowite o stałym rozmiarze

- Od standardów *C99* i *C++11* wprowadzono typy całkowite o stały rozmiarze.
- Rozwiązanie to zwiększa uniwersalność kodu w sytuacji różnych architektur, platform czy zastosowanego kompilatora.
- Plik nagłówkowy **stdint.h** dla języka C, **cstdint** dla języka C++.

Słowo kluczowe	Rozmiar [B]	Zakres wartości
<code>int8_t</code>	1	[-128; +127]
<code>int16_t</code>	2	[-32768; +32767]
<code>int32_t</code>	4	[-2147483648; +2147483647]
<code>int64_t</code>	8	[-9223372036854775808; +9223372036854775807]
<code>uint8_t</code>	1	[0; 255]
<code>uint16_t</code>	2	[0; 65535]
<code>uint32_t</code>	4	[0; 4294967295]
<code>uint64_t</code>	8	[0; 18446744073709551615]

Typy proste

```
# include <stdio.h>

int main () {
    printf ("\nSize of short: %d", sizeof (short));
    printf ("\nSize of short int: %d", sizeof (short int));
    printf ("\nSize of int: %d", sizeof (int));
    printf ("\nSize of long : %d", sizeof (long));
    printf ("\nSize of long int : %d", sizeof (long int));
    printf ("\nSize of long long : %d", sizeof (long long));
    printf ("\nSize of float: %d", sizeof (float));
    printf ("\nSize of double: %d", sizeof (double));
    printf ("\nSize of long double: %d", sizeof (long double));
    printf ("\n");
    return 0;
}
```

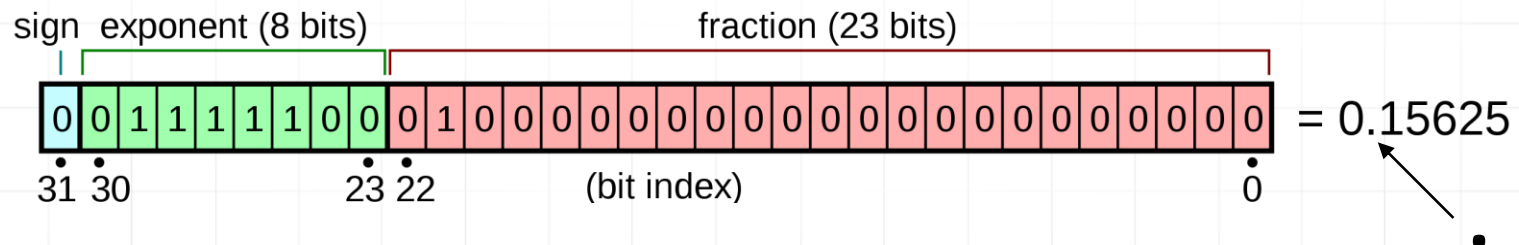
```
Size of short: 2
Size of short int: 2
Size of int: 4
Size of long : 4
Size of long int : 4
Size of long long : 8
Size of float: 4
Size of double: 8
Size of long double: 12
```

Typy proste - liczby

Liczba całkowita – jest liczbą nieposiadającą części ułamkowej. W języku C/C++ liczby całkowite nie zawierają kropki dziesiętnej. Liczby całkowite są przechowywane w postaci binarnej.

Liczba zmiennoprzecinkowa – liczba rzeczywista, może posiadać część ułamkową. Zapis liczby zmiennoprzecinkowej składa się z dwóch części: binarnego ułamka i binarnego wykładnika

$$x = (-1)^S \cdot M \cdot 2^{E-bias}$$



<https://wikipedia.org/>

Systemy liczbowe

System dziesiętnym	System binarny	System ósemkowy	System szesnastkowy
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

Systemy liczbowe

- Możliwości inicjalizowania zmiennej z wykorzystaniem różnych systemów liczbowych:

```
int a = 180; // system dziesiętny
int b = 0b10110100; // system binarny (prefiks 0b)
int c = 0264; // system osemkowy (prefiks 0)
int d = 0xB4; // system szesnastkowy (prefiks 0x)
```

Funkcja printf() – wyświetlanie wartości danych

Specyfikatory formatu :

- %c** pojedynczy znak
- %s** łańcuch znaków
- %d** liczba dziesiętna ze znakiem
- %f** liczba zmiennoprzecinkowa (notacja dziesiętna)
- %e** liczba zmiennoprzecinkowa (notacja wykładnicza)
- %g** liczba zmiennoprzecinkowa (krótszy z formatów %f %e)
- %u** liczba dziesiętna bez znaku
- %x** liczba w kodzie szesnastkowym (bez znaku)
- %o** liczba w kodzie ósemkowym (bez znaku)

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

```
int main( )
```

```
{
```

```
    int x = 100;
```

```
    int y = 200;
```

```
    double d;
```

```
    d = x + y;
```

```
    printf ( "Suma obliczen %d + %d = %f", x, y, d);
```

```
}
```

```
Suma obliczen 100 + 200 = 300.000000
```

Tekst sterujący

Argumenty

Rodzaje zmiennych

Zmienne:

- **lokalne (automatyczne)** – są to zmienne/obiekty zadeklarowane wewnątrz bloku, ciała funkcji i są dostępne wyłącznie w tym bloku. Czas życia **zmiennej lokalnej** zaczyna się od chwili wejścia sterowania programu w dany blok (od miejsca definicji) do chwili wyjścia z niego.
- **globalne (zewnętrzne)** – są to zmienne deklarowane poza wszystkimi funkcjami i są dostępne dla wszystkich funkcji. Czas życia to czas uruchomienia programu.

Przykład

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

```
int z=0;           // z zmienna globalna
```

```
void f() {
```

```
    int x=0;       // x zmienna lokalna (automatyczna)
```

```
    ...
```

```
}
```

```
int main() {       // i zmienna lokalna (automatyczna)
```

```
    for(int i=0; i<3; ++i){
```

```
        f();
```

```
    }
```

```
    return 0;
```

```
}
```

Rodzaje zmiennych – ze względu na czas życia (storage class)

Statyczne (`static`)

- Zmienna o statycznym przydziel pamięci. Pamięć jest zarezerwowana na stałe w trakcie działania programu. **Czas życia** od momentu inicjalizacji do końca programu.
- Deklarujemy za pomocą słowa kluczowego `static`
np. `static int x;`
- Inicjalizowane wartościami początkowymi tylko raz, przed rozpoczęciem wykonania programu. Jeżeli zmienna statyczna nie zostanie jawnie zainicjalizowane, wówczas standardowo nadana zostanie jej wartość 0.

Automatyczne (`auto`)

- Domyślnie zmienne lokalne, zadeklarowane wewnątrz funkcji lub wewnątrz instrukcji blokowej, gdy nie są poprzedzone słowem kluczowym `static`.
- Parametry/argumenty funkcji są również zmiennymi automatycznymi.
- Czas życia jest związany z wykonaniem instrukcji blokowej, między `{}`. Pamięć dla zmiennych przydzielana jest w momencie wejścia do instrukcji blokowej (od miejsca definicji) i zwalniana w momencie wykonania ostatniej instrukcji bloku.
- Jeżeli nie zainicjujemy wartością początkową, wówczas będzie ona miała wartość nieokreśloną.

Czas życia – typowy przykład

```
#include <stdio.h>

int z=0;                // zmienna globalna

void f() {
    int x=0;            // zmienna lokalna (automatyczna)
    static int y=0;     // zmienna statyczna
    ++x;
    ++y;
    z=z+2;
    printf("x=%d, y=%d, z=%d \n", x, y, z);
}

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

Jaki będzie rezultat ?

Czas życia – typowy przykład

```
#include <stdio.h>

int z=0;                // zmienna globalna

void f() {
    int x=0;            // zmienna lokalna (automatyczna)
    static int y=0;     // zmienna statyczna
    ++x;
    ++y;
    z=z+2;
    printf("x=%d, y=%d, z=%d \n", x, y, z);
}

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

```
x=1, y=1, z=2
x=1, y=2, z=4
x=1, y=3, z=6
```

Zakres widoczności

Scope czyli **zakres widoczności** zmiennych/funkcji jest to obszar programu, w którym można się odwoływać do danego identyfikatora funkcji lub zmiennej. Wyróżnia się następujące obręby widoczności:

- Wnętrze **pliku** – deklaracja zmiennej poza definicją funkcji lub listą jej parametrów, widoczna od momentu deklaracji do końca pliku (jednostki translacji)
- Wnętrze **funkcji** – widoczność etykiety instrukcji (`goto etykieta;`)
- Wnętrze **bloku** instrukcji – definicja zmiennej wewnątrz ciała bloku pomiędzy nawiasami { } lub listy argumentów funkcji. Blokiem instrukcji możemy nazwać np. definicję funkcji, pętle i wyrażenia warunkowe wykorzystujące {}.

Rodzaje zmiennych – ze względu na czas życia

Porównanie rodzajów zmiennych w językach C/C++

Rodzaj zmiennej	Zasięg (widoczność)	Rodzaj wiązania	Czas życia	Domyślna inicjalizacja	Segment pamięci
lokalna	w obrębie bloku	brak	w obrębie bloku	brak	stos
globalna	cały program	zewnętrzne	cały program	zerowanie	.data ¹
statyczna lokalna	w obrębie bloku	brak	cały program ²	zerowanie	.data ¹
statyczna globalna	w obrębie jednostki translacji	wewnętrzne	cały program	zerowanie	.data ¹

¹Dotyczy zainicjalizowanych zmiennych

²Od momentu inicjalizacji do końca działania programu

Rodzaje zmiennych – ze względu na czas życia

Porównanie rodzajów zmiennych w językach C/C++

Rodzaj zmiennej	Zasięg (widoczność)	Rodzaj wiązania	Czas życia
lokalna	w obrębie bloku	brak	w obrębie bloku
globalna	cały program	zewnątrzne	cały program
statyczna lokalna	w obrębie bloku	brak	cały program ²
statyczna globalna	w obrębie jednostki translacji	wewnętrzne	cały program

- Utworzenie dwóch zmiennych globalnych o tej samej nazwie, w różnych jednostkach translacji (plikach źródłowych) spowoduje zgłoszenie przez kompilator błędu wielokrotnej definicji.
- Ale, w kompilacji rozłącznej, aby odwołać się do zmiennej globalnej z innej jednostki translacji trzeba użyć słowa kluczowego `extern` (storage class).

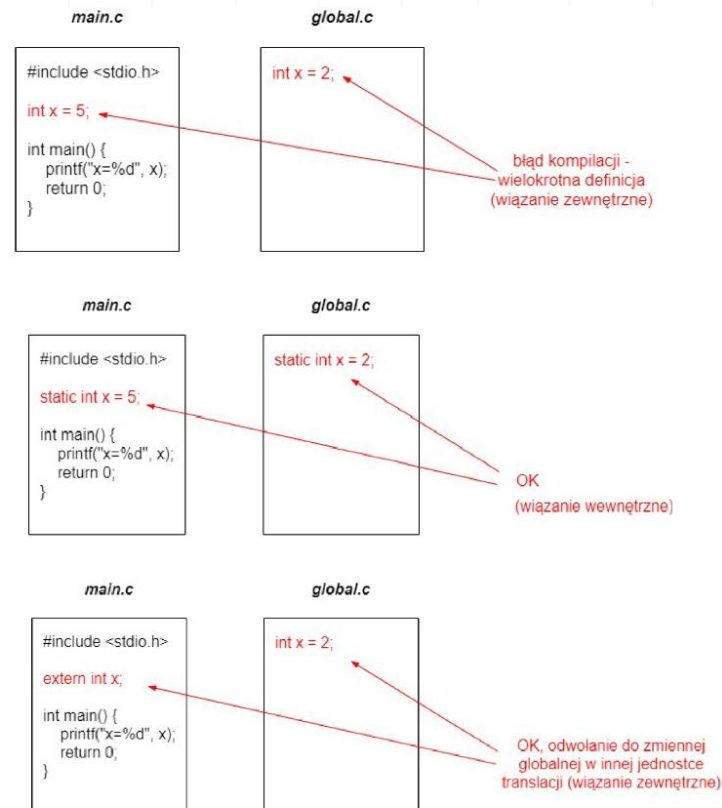
¹Dotyczy zainicjalizowanych zmiennych

²Od momentu inicjalizacji do końca działania programu

Rodzaje zmiennych – ze względu na czas życia

Porównanie rodzajów zmiennych w językach C/C++

Rodzaj zmiennej	Zasięg (widoczność)	Rodzaj wiązania	Czas życia
lokalna	w obrębie bloku	brak	w obrębie bloku
globalna	cały program	zewnątrzne	cały program
statyczna lokalna	w obrębie bloku	brak	cały program ²
statyczna globalna	w obrębie jednostki translacji	wewnętrzne	cały program



- Specyfikator **extern** mówi, że zmienna jest zdefiniowana gdzie indziej, nie w tym samym bloku. Główne zastosowanie – dostęp do zmiennych globalnych zdefiniowanych w różnych plikach

M. Stępnia, Laboratorium Informatyki

Zakres widoczności – mechanizm przysłaniania

```
#include <iostream>

int a; // zasięg globalny

int main(){
    printf("Zmienna globalna a = %d\n", a);
    printf("Definicja nowej zmiennej lokalnej a\n");
    int a = 1; // zasięg lokalny 1
    printf("Zasięg lokalny 1, a = %d\n", a);

    { //Przysłanianie zmiennych w języku C
        printf("{\n\tZasięg lokalny 2, a = %d\n", a);
        printf("\tDefinicja nowej zmiennej lokalnej a\n");
        int a = 2; // zasięg lokalny 2
        printf("\tZasięg lokalny 2, a = %d\n", a);

        {
            printf("\t{\n\t\tZasięg lokalny 3, a = %d\n", a);
            printf("\t\tDefinicja nowej zmiennej lokalnej a\n");
            int a = 3; // zasięg lokalny 3
            printf("\t\tZasięg lokalny 3, a = %d\n", a);

            {
                printf("\t\t{\n\t\t\t");
                extern int a; // zasięg lokalny 4
                printf("\t\t\tOdwolanie sie do zmiennej globalnej"
                    " wewnatrz zasięgu lokalnego 4 ( język C), a = %d\n", a);
                printf("\t\t\t");
            }

            printf("\t\tZasięg lokalny 3, a = %d\n", a);
            printf("\t\tOdwolanie sie do zmiennej globalnej"
                " wewnatrz zasięgu lokalnego 4 ( język C++), a = %d\n", ::a);
            printf("\t\t");
        }

        printf("\tZasięg lokalny 2, a = %d\n", a);
        printf("\t");
    }

    printf("Zasięg lokalny 1, a = %d\n", a);
    return 0;
}
```

Zakres widoczności – mechanizm przysłaniania

```
#include <iostream>

int a; // zasięg globalny

int main(){
    printf("Zmienna globalna a = %d\n", a);
    printf("Definicja nowej zmiennej lokalnej a\n");
    int a = 1; // zasięg lokalny 1
    printf("Zasięg lokalny 1, a = %d\n", a);

    { //Przysłanianie zmiennych w języku C
        printf("\n\tZasięg lokalny 2, a = %d\n", a);
        printf("\tDefinicja nowej zmiennej lokalnej a\n");
        int a = 2; // zasięg lokalny 2
        printf("\tZasięg lokalny 2, a = %d\n", a);

        {
            printf("\t\t\n \t\tZasięg lokalny 3, a = %d\n", a);
            printf("\t\tDefinicja nowej zmiennej lokalnej a\n");
            int a = 3; // zasięg lokalny 3
            printf("\t\tZasięg lokalny 3, a = %d\n", a);

            {
                printf("\t\t\t\n");
                extern int a; // zasięg lokalny 4
                printf("\t\t\tOdwolanie sie do zmiennej globalnej"
                    " wewnatrz zasięgu lokalnego 4 ( język C), a = %d\n", a);
                printf("\t\t\t\n");
            }

            printf("\t\tZasięg lokalny 3, a = %d\n", a);
            printf("\t\tOdwolanie sie do zmiennej globalnej"
                " wewnatrz zasięgu lokalnego 4 ( język C++), a = %d\n", ::a);
            printf("\t\t\n");
        }

        printf("\tZasięg lokalny 2, a = %d\n", a);
        printf("\n");
    }

    printf("Zasięg lokalny 1, a = %d\n", a);
    return 0;
}
```

extern (Język C)

- VS.-

:: (Język C++)

```
Zmienna globalna a = 0
Definicja nowej zmiennej lokalnej a
Zasięg lokalny 1, a = 1
{
    Zasięg lokalny 2, a = 1
    Definicja nowej zmiennej lokalnej a
    Zasięg lokalny 2, a = 2
    {
        Zasięg lokalny 3, a = 2
        Definicja nowej zmiennej lokalnej a
        Zasięg lokalny 3, a = 3
        {
            Odwolanie sie do zmiennej globalnej
            wewnatrz zasięgu lokalnego 4 ( język C), a = 0
        }
        Zasięg lokalny 3, a = 3
        Odwolanie sie do zmiennej globalnej
        wewnatrz zasięgu lokalnego 4 ( język C++), a = 0
    }
    Zasięg lokalny 2, a = 2
}
Zasięg lokalny 1, a = 1
```

Zmienne globalne – zalety, wady

- Należy ograniczać stosowanie **zmiennych globalnych** w kodzie aplikacji (jeżeli to możliwe stosować wiązane wewnętrzne `static`), ze względu (wady):
 - mogą być modyfikowane w każdej funkcji lub wątku,
 - są przesłanianie przez zmienne lokalne.
- Zalety wiążą się z stosowanie ich dla prostych architektur mikrokontrolerów o małej pamięci stosu oraz dla procesorów niespierających dynamicznej alokacji pamięci.