Programowanie Niskopoziomowe

Wykład 1

- Wprowadzenie

Instytut Informatyki

dr inż. Marcin Radom

Informacje organizacyjne

- Wykład – zasady zaliczenia
- Literatura
- Plan wykładów

Zasady zaliczenia

- **Laboratorium**
  - Zaliczenie ćwiczeń na podstawie oceny z projektu zaliczeniowego oraz kolokwium (**lub jakkolwiek ustali prowadzący**).

- Wykłady dostępne są na platformie e-kursy

- Wykład i laboratorium są niezależne.

- **Ocena z wykładu** – na podstawie kolokwium zaliczeniowego w czerwcu.

Literatura

- Brian W. Kernighan, Dennis R. Ritchie, **ANSI C**, seria *Klasyka Informatyki*.
- Jerzy Grębosz, *Symfonia C++*, Oficyna Kallimach, Kraków, 1999.
- Stephen Prata, *Szkoła Programowania Język C++*, Robomatic 2002.

- Liczne opracowania dostępne w Internecie. A w praktyce jakikolwiek sobie Państwo podręcznik znajdziecie, tych związanych z C, starych i nowych, nie zliczy nikt.

Plan wykładów

- **Wykład I**
  - Wprowadzenie do języka C: historia, identyfikatory, alfabet
  - Typy, stałe
- **Wykład II**
  - Instrukcja warunkowa **if-else**
  - Instrukcja wyboru **switch**
  - Instrukcje pętli **for**, **while**, **do while**
  - Instrukcje **break**, **continue**
  - Preprocesor: **#define** i **makrogeneracja**, kompilacja warunkowa
- **Wykład III**
  - Podstawowe operatory, operatory bitowe, warunkowy, wyliczenia,
  - Priorytety i łączność operatorów
  - Przekształcenia typów danych
  - Wprowadzanie (m. in. **scanf**) i wyprowadzanie danych (m.in. **printf**)
- **Wykład IV**
  - Tablice jedno i wielowiarowe
  - Przetwarzanie tekstów (biblioteka string.h)

Plan wykładów

- **Wykład V**
  - Dynamiczny przydział pamięci, tablice wskaźników
  - Struktury i unie
  - Struktury dynamicznej – dynamiczny przydział pamięci, lista
  - Tablice struktur, wskaźniki do struktur
  - Łączenie wielu plików programu - podział programu na moduły
  - Zmienne statyczne
- **Wykład VI**
  - Funkcje – definicja, wywołanie, sposoby przekazywania argumentów do funkcji
  - Struktury i tablice jako argumenty funkcji
  - Rekurencja
- **Wykład VII**
  - Wskaźniki do funkcji
  - Tablice wskaźników do funkcji
  - Pliki dyskowe – odczyt i zapis danych

Historia języka C

- Historia, cechy, popularność

Historia języka C

- Rozwijany w latach **1969-1973** (napisano w nim jądro systemu UNIX w roku 1973).
- Poprzednikiem był język **B** (**1969**, opracowany przez Kena Thompsona dla pierwszego systemu Unix), którego poprzednikiem był **BCPL** (**1966**, opracowany przez Martina Richardsa)
- 1978: *The C Programming Language*, Brian Kernighan & Dennis Ritchie podręcznik języka, służący jako jego nieformalna specyfikacja.
- Rozszerzenie obiektowe C to **C++** (rozszerzono też część nie obiektową).
- W **1983 A**merykański **N**arodowy **I**nstytut **S**tandaryzacji (**ANSI**) powołał komitet, którego zadaniem było sformułowanie wszechstronnej definicji języka C.
- W **1989** powstało opracowanie tzw. standardu ANSI lub inaczej **ANSI C**, współczesne kompilatory realizują większość cech tego standardu.
  - Język uległ niewielkim zmianom np. zdefiniowano bibliotekę towarzyszącą C (np. czytanie/zapis do pliku etc.), w deklaracjach funkcji wprowadzono opis parametrów, opracowano możliwości preprocesora (wstępna faza kompilacji, włączanie do programu zawartości innych plików źródłowych oraz kompilacja warunkowa), struktury i wyliczenia stały się częścią języka etc.
  - Założeniem standardu było zagwarantowanie poprawności istniejących już programów.

Historia języka C

- W roku **1990** standard języka C został zapisany w normie **ISO 9899**. Wydanie tego dokumentu było modyfikacją standardu ANSI. Język zgodny z tą wersją standardu określany jest nieformalnie jako **C89**.

- Od tego czasu powstało wiele uaktualnień. Najważniejsza wersja to **ISO 9899:1999**, a język z nią zgodny określany jest nieformalnie **C99**. Wprowadziła ona istotne zmiany w stosunku do **ANSI C**, np.:
  - Dopuszczalne są komentarze w stylu C++ obejmujące pojedynczą linię //
  - Nowe standardowe funkcje i pliki nagłówkowe
  - Rozszerzono aparat preprocesora
  - Wprowadzono nowe słowa kluczowe np. *const, enum, signed, void*
  - Możliwość deklarowania zmiennych w dowolnym miejscu programu.

- Standard **ISO 9899:1999** nie jest w pełni wspierany przez dostępne kompilatory (szczególnie przez *Microsoft Visual C++*). Kompilator GNU C (podstawa DevC++) uwzględnia większość zmian.

Historia języka C

- Najnowszy standard to **ISO/IEC 9899:2011**, określany nieformalnie jako **C11.** Wprowadza m.in. następujące zmiany:
  - Wsparcie wielowątkowości.
  - Dodano standardowe pliki nagłówkowe, np.: *<threads.h>, <stdatomic.h>, <uchar.h>*.
  - Anonimowe struktury oraz unie (użyteczne w zagnieżdżonych definicjach np. unia będąca składową struktury).
  - Nowe słowa kluczowe np.: *_Generic, _Thread_local, _Alignas*
  - Wparcie zapisu znaków wg norm **Unicode**, m.in. poprzez dodanie nowych typów danych niezależnych od platformy: *char16_t, char32_t*
  - Bezpieczniejsze odpowiedniki istniejących funkcji - dodatkowo wymagają, na ogół, argumentu określającego wielkość przetwarzanego bufora (z *_s* na końcu np. *strcat_s()*)).
  - Funkcja *gets()* została usunięta i zastąpiona przez *gets_s()*
  - Dostępna jest bezpieczniejsza wersja funkcji *fopen* o nazwie *fopen_s*
- Kompilator GCC wspiera standard C11 w ograniczonym zakresie. Aby skompilować kod zgodnie z tym standardem należy wybrać opcję kompilatora: **-std=c11** lub **-std=iso9899:2011**

Historia języka C

- **C (ANSI C)** jest językiem strukturalnym, poprzednikiem współcześnie używanych języków obiektowych.
- Jest to **proceduralny język programowania** wyposażony w podstawowe konstrukcje sterujące:
  - grupowanie instrukcji, podejmowanie decyzji (**if**-**else**),
  - wybór jednego ze zbioru możliwych przypadków (**switch**),
  - powtarzanie ze sprawdzaniem warunku na początku (**while**, **for**) lub na końcu (**do**) pętli
  - przerwanie pętli (**break**).
- Określany jest jako język „niskiego poziomu”, ponieważ posługuje się znakami, liczbami oraz adresami, a nie obiektami złożonymi. Obiekty te mogą być łączone lub przemieszczane za pomocą zwykłych operacji arytmetycznych i logicznych.
- Język C jest językiem ogólnego stosowania (nie jest przeznaczony dla żadnej szczególnej dziedziny zastosowań).
- Język C jest przenośny, gdyż nie jest przywiązany do żadnego systemu operacyjnego lub maszyny (może być uruchamiany bez zmian na różnorodnym sprzęcie).

Programowanie w C

- Identyfikatory, typy, zmienne, stałe

Prosty przykład

```
#include <stdio.h>
int main( ) {
        printf ( "Hello World!\n" );
        return 0;
}
```

- **#include <stdio.h>** - polecenie dla preprocesora, aby dołączył informację o podstawowej, standardowej bibliotece wejścia-wyjścia: *stdio* – **ST**an**D**ard **I**nput **O**utput.
- Funkcja **main()** – program zawsze rozpoczyna działanie od początku funkcji **main**. W przykładzie jest bezargumentowa (tj. puste nawiasy, niezbędne wg składni języka), czyli nie oczekuje żadnych argumentów.
- Wewnątrz funkcji **main()** wywołujemy biblioteczną funkcję **printf**("Hello World!\n"), która na ekranie wyświetla zadany napis. **\n** reprezentuje znak nowego wiersza.

- Znak \ wprowadza zaraz po sobie jakiś **znak specjalny**, niemożliwy do bezpośredniego wpisania. Przykładowe sekwencje w języku C: **\t** dla znaku tabulacji, **\b** dla znaku cofania kursora o 1 pozycję, **** dla znak \, **\"** dla znak cudzysłowu (nie można go wprost uzyskać na ekranie, ponieważ jest znakiem zastrzeżonym).

Prosty przykład

- Funkcja **printf** automatycznie nie wstawi znaku nowej linii, więc jeśli go zabraknie, kolejne wywołania funkcji **printf** będę 'budować' ten sam wiersz na ekranie. Na przykład:

```
printf("Hello ");
printf("World!");
printf("\n");
```

dadzą w rezultacie identyczny wynik na ekranie, co pojedyncze wywołanie funkcji **printf( )** pokazane na poprzednim slajdzie.

- Nawiasy klamrowe otaczają instrukcje funkcji.

- **Przypomnienie:**
  - Zmienne i stałe są podstawowymi obiektami danych, jakimi posługuje się program.
  - Deklaracje wprowadzają potrzebne zmienne oraz ustalają ich typy i ewentualnie wartości początkowe.
  - Operatory określają co należy z nimi zrobić.
  - Wyrażenia wiążą zmienne i stałe, tworząc nowe wartości.
  - Typ obiektu determinuje zbiór jego wartości i operacje, jakie można na nim wykonać.

Alfabet języka C

- Alfabet C - zbiór znaków, za pomocą których zapisuje się programy w języku C.

- **Zawiera:**
  - Wszystkie znaki 8-bitowego kodu ASCII, czyli:
    - Duże litery alfabetu łacińskiego: A B … Z
    - Małe litery alfabetu łacińskiego: a b … z
    - Cyfry: 1 … 9
    - Znaki specjalne: ! * + \ " < # (= | { > %) ~ ; } / ^ - [: , ? & _] ' oraz znak odstępu (spacja)

- Język C (dokładniej wersja C99) wspiera zapis znaków wg norm **Unicode** (uniwersalny standard kodowania znaków, dzięki któremu można wyświetlać znaki charakterystyczne dla różnych języków).
  - Kod nie będzie wtedy zgodny ze starszymi wersjami standardu ANSI C, co zmniejsza jego przenośność.
  - Najczęściej stosowany sposób kodowania to UTF-8.

Identyfikatory, rozkazy, typy

- Nazwa zmiennej może zawierać litery i cyfry, musi jednak zaczynać się od litery (znak _ zaliczany jest do liter).

- W języku C rozróżnia się małe i duże litery alfabetu, więc *zmienna* i *Zmienna* to dwie różne nazwy. Inne przykłady:

```
alfa      Alfa      AlfA      ALFA
Cena_Mleka     KosztTransportu
```

- Zazwyczaj przyjmuje się, że nazwy zmiennych piszemy małymi, a nazwy stałych symbolicznych dużymi literami.

- **Słowa kluczowe** języka są zarezerwowane i nie można ich używać jako nazw zmiennych. Przykładowe słowa kluczowe języka C:
  - **auto** – archaiczne oznaczenie zmiennej lokalnej
  - **double** – typ rzeczywisty podwójnej dokładności
  - **int** – typ całkowity ze znakiem
  - **struct** – deklaracja struktury (odpowiednik rekordu)
  - **break** – wyjście z pętli lub instrukcji wyboru
  - **else** – opcjonalna część instrukcji warunkowej *if*
  - **long** – modyfikator lub/i typ danych
  - **switch** – instrukcja wyboru

Identyfikatory, rozkazy, typy

- **case** – alternatywa w instrukcji wyboru
- **enum** – deklaracja typu wyliczeniowego
- **register** - klasa pamięci, zmienna rejestrowa. Prośba do kompilatora, żeby trzymał zmienną w rejestrze procesora. Niezalecane.
- **typedef** – nazywanie typów
- **char** – typ znakowy
- **extern** – zmienna globalna zadeklarowana w innym pliku; symbol zewnętrzny. Informacja dla kompilatora, żeby nie szukał definicji zmiennej w danym pliku.
- **return** – instrukcja powrotu z funkcji (podprogramu)
- **union** – deklaracja unii (odpowiednik rekordu z wariantami)
- **const** – kwalifikator typu, wartość nie będzie modyfikowana
- **float** – typ rzeczywisty pojedynczej precyzji
- **short** – modyfikator i/lub typ danych
- **unsigned** – modyfikator, zmienna bez znaku

Identyfikatory, rozkazy, typy

- **continue** – instrukcja powrotu do początku pętli
- **for** – instrukcja, część pętli *for*
- **signed** - modyfikator, zmienna ze znakiem
- **void** – typ danych
- **default** – domyślna alternatywa w instrukcji wyboru
- **goto** – instrukcja skoku
- **sizeof** – operator rozmiaru
- **volatile** – zmienna jest zawsze czytana z pamięci
- **do** – część pętli *do-while*
- **if** – element instrukcji warunkowej *if*
- **static** – wartość zmiennej jest zachowywana pomiędzy kolejnymi wywołaniami bloku; zmienna statyczna/symbol lokalny, np. gdy zdefiniujemy zmienną statyczną w ciele funkcji to jej wartość nie zmieni się przy ponownym wywołaniu funkcji.
- **while** – część pętli *while* i *do-while*

Komentarze

- **Komentarze**
  - Znaki po **//** są traktowane jako początek komentarza 1-linijkowego (komentarz w stylu C++)
    - **Np.:**

```
instrukcja;  // komentarz, tu wpisać cokolwiek
// ..............................
```

  - **/*** i ***/** to znaki otwierające/zamykające komentarz który może rozciągać się na wiele linii.

```
/*
      ......................
      bez zagnieżdżania
      ......................
*/
```

  - Komentarze nie mogą być zagnieżdżane (standard ANSI C).
  - Nie mogą wystąpić w napisach i stałych znakowych.

Typy liczb całkowitych

- W języku C występuje kilka podstawowych typów danych:
  - Typy liczb całkowitych

typ	signed	unsigned	bajty
`char`	– 128 , + 127	0 , 255	1
`short`	– 32 768 , + 32767	0 , 65535	2
`int`, `long`	– 2 147 483 648 , + 2 147 483 647	0 , 4 294 967 295	4
`long long`	– 9 223 372 036 854 775 808 , + 9 223 372 036 854 775 807	0 , 18 446 744 073 709 551 615	8

  - Dodatkowo występuje kilka **kwalifikatorów** stosowanych razem z typami liczb całkowitych:
    - **short** oraz **long** wprowadzono po to, aby umożliwić posługiwanie się różnymi zakresami liczb całkowitych.

Typy liczb całkowitych

- Typ **int** na ogół odzwierciedla rozmiar wynikający z architektury danej maszyny.
- Obiekt typu **short** często zajmuje **2 bajty**, a obiekt typu **long** **4 bajty.**
- **Kompilator może dowolnie wybierać rozmiary typów w zależności od sprzętu na jakim pracuje, ale musi zachować pewne ograniczenia:**
  - Typy **short** oraz **int** muszą być co najmniej 2-bajtowe.
  - Typy **long** muszą być co najmniej 4-bajtowe.
  - Obiekt **short** nie może być dłuższy niż **int**, a **int** nie może być dłuższy niż **long**.
- Kwalifikatory **signed** (ze znakiem) i **unsigned** (bez znaku) można stosować razem z typem **char** lub dowolnym typem całkowitym.
- Liczby **unsigned** są zawsze dodatnie lub równe zero i podlegają regułom arytmetyki modulo 2^n (n – liczba bitów reprezentująca dany typ), np. zmienna typu **unsigned char** będzie przyjmować wartości z zakresu od 0 do 255.
- `char  short  int  long  long long  signed  unsigned`

Typy zmiennopozycyjne

- **Typy zmiennopozycyjne:**

typ	zakres	bajty
`float`	$\pm 3.4 * 10^{\pm 38}$	4
`double`, `long double`	$\pm 1.7 * 10^{\pm 308}$	8

  - Typ **long double** wprowadza liczbę zmiennopozycyjną o rozszerzonej precyzji.
  - Rozmiary obiektów zmiennopozycyjnych (podobnie jak dla typów całkowitych) zależą od implementacji.
  - Typy **float**, **double**, **long double** mogą reprezentować jeden, dwa lub trzy różne rozmiary obiektów.

Liczby zmiennopozycyjne

- Standard zapisu zmiennoprzecinkowego jest zgodny ze standardem **IEEE 754.** Definiuje on dwie podstawowe klasy binarnych liczb zmiennoprzecinkowych:
  - 32-bitowa (pojedyncza precyzja)
  - 64-bitowa (podwójna precyzja)

Format	Bit znaku	Bit cechy	Bit mantysy
32 bity	1	8	23
64 bity	1	11	52

- Gdzie liczbę zmiennopozycyjną zapisujemy:
  - Liczba cyfr mantysy decyduje o dokładności zmiennopozycyjnego przedstawienia liczb, a liczba cyfr cechy określa zakres reprezentowanych liczb.

$$D_{FP} = mantysa * 2^{cecha}$$

Wartości typu **float / double**

- Liczby zmiennopozycyjne zapisujemy z kropką dziesiętną (np. 120.4) lub z wykładnikiem (potęgi) : np. 1e-2, lub też obie te części występują naraz.

- Przykładowe wartości:

```
1.25        0.343      .5       2.
35.56E-12    0.34e2     5e3     17.18E+28
```

- **Typ liczby określany jest:**
  - Na podstawie wartości (domyślnie **double**)
  - Wskazany w zapisie liczby
    - Występująca na końcu litera ***f*** lub ***F*** oznacza stałą typu **float**, a litera ***l*** lub ***L*** stałą typu **long double**. Np.:

```
12.545f                 // float
0.2345676543F           // float
0.5e-3l1f               // long double
0.9999998899E456LF      // long double
```

Wartości całkowite (typu **int**)

- **Typ liczby określany jest:**
  - Na podstawie wartości (domyślnie **int**)
    - Jeśli zmienna bez *L* na końcu nie mieści się w **int**, będzie potraktowana jako typu **long**. Na przykład:

```
12    25467          // signed int
34760548093          // signed long long
```

  - Wskazany w zapisie liczby
    - W zmiennej typu **long** na końcu występuje litera ***l*** (małe "L"!) lub ***L***.
    - W zmiennych typu **unsigned** na końcu występuje litera ***u*** lub ***U***. Końcówka ***ul*** oraz ***UL*** oznacza stałą typu **unsigned long**.
    - Przykładowe wartości:

```
15L    077777l   0xFF4FFFL      // signed long
254ll     -457LL   0xAB56LL     // signed long long
45211u    0xffau                // unsigned int
3000000000ul   0xC56AFB44UL     // unsigned long
-120ULL  78ull                  // unsigned long long
```

Zmienne znakowe

- Wartość całkowitą można przedstawić również w postaci ósemkowej lub szesnastkowej:
  - **0** (zero) przed zapisaniem liczby oznacza zapis ósemkowy, np. 077 to 63 dziesiętne (77 ósemkowe)
  - Zapis szesnastkowy jest realizowany poprzez dodanie **0x** lub **0X** na początku, np. 0xFF = 0XFF (255 dziesiętnie)
  - Przykładowe wartości:

```
12    154555             // dziesiętnie
012   03777453           // ósemkowo
0xAB  0x5c5d   0xfff45a  // szesnastkowo
```

- **Stałe znakowe czyli liczby całkowite typu: `char`**
  - Pojedyncza zmienne znakowa jest liczbą całkowitą, mającą wartość od 0 do 255. Jest to podyktowane tym, że znak graficzny jest zapisywany najczęściej pojedynczą liczbą szesnastkową.
  - Taką zmienną tworzy znak ujęty w apostrofy, np. 'x'.
  - Pewne znaki niegraficzne mogą być reprezentowane w zmiennych znakowych i napisowych przez sekwencje/znaki specjalne, takie jak **\n** (znak nowego wiersza).

Stałe znakowe

- Znaki specjalne (\n, \t, itd.) również można zapisać jako stałą znakową przy użyciu zapisu ósemkowego lub szesnastkowego. Np.:

```
#define VTAB '\013'          //ASCII: pionowy tabulator
#define BELL '\007'          //ASCII: 'znak' alarmu
#define VTAB '\xb'           //ASCII: pionowy tabulator
#define BELL '\x7'           //ASCII: 'znak' alarmu
```

- **Lista sekwencji specjalnych języka C:**

\a	alarm	****	znak \
\b	znak cofania	**\?**	znak zapytania
\f	znak nowej stopy	**\'**	znak apostrofu
\n	nowy wiersz	**\"**	cudzysłów
\r	powrót karetki(CR)*	**\ooo**	liczba ósemkowa
\t	tabulacja pozioma	**\xff**	liczba szesnastkowa
\v	tabulacja pionowa		

  - *CR – *carriage return* – 'powrót karetki' – z czasów maszyny do pisania, cofnięcie kursora do początku linii, często jako kombinacja \n\r - czyli nowa linia, od lewej.

Łańcuchy

- Przykładowe zmienne znakowe:

```
'a'         '5'         '+'         '.'
'A'         '\071'      '\x41'      '\x5F'
'\n'        '\t'        '\r'        '\\'        '\"'
```

- Stała znakowa **'\0'** reprezentuje znak o wartości zero, tzw. znak pusty. Stosuje się ją zamiast liczby 0 dla podkreślenia znakowej natury pewnych wyrażeń.

- **Wyrażenie stałe** :

```
#define LEAP 1
int tab[31+30+LEAP+28]
```

- **Stała napisowa (napis, łańcuch)** jest ciągiem złożonym z zera lub więcej znaków, zawartym między znakami cudzysłowu, np.: `"jestem napisem"` albo `""` (napis pusty).
  - Znaki " " nie są częścią napisu, określają tylko jego początek i koniec, jeśli taki znak MA być częścią napisu, należy użyć \" wewnątrz cudzysłowów, np. " to jest cudzysłów: \" "
  - Napisy mogą być sklejane podczas kompilacji kodu (przydatne podczas dzielenia długich napisów na wiersze w kodzie źródłowym).

Łańcuchy

- Przykładowe łańcuchy:

```
"Programowanie w języku C"
"Wynik : "
"\tImię\tNazwisko\tMiejsce zamieszkania\n"
"\x16\x16\x02"          // SYN SYN STX
"Spojrzał i powiedział: \"Nie wiem\"."
```

- Stała napisowa jest tablicą, której elementami są znaki.

- Wewnętrzna reprezentacja napisu zawiera na końcu znak **'\0'** stąd rozmiar pamięci fizycznej przeznaczonej na napis jest o jeden większa niż liczba znaków zawartych pomiędzy znakami cudzysłowu. Np.:

"ABC" **:**

0x41	**0x42**	**0x43**	**0x00**

- Taka reprezentacja oznacza, że nie ma ograniczenia dotyczącego długości tekstów.

Łańcuchy

- Programy muszą badać tekst, żeby określić jego długość.
- Przydatna funkcja: **strlen(napis)** - zwraca długość napisu, **bez znaku końcowego**, czyli bez wliczania '\0'.
- Funkcja **strlen** oraz inne funkcje wykonujące operacje na tekstach są zadeklarowane w standardowym pliku nagłówkowym **<string.h>** (aby ich użyć należy dołączyć do programu #include <string.h>).
- Różnica pomiędzy znakami ' ' oraz " ":
  - 'c' to pojedynczy znak, mający wartość określającą jego położenie w tablicy znaków ASCII
  - "c" to tablica znaków, zawierająca tak naprawdę dwa znaki: literę c oraz niejawny znak \0 oznaczający koniec ciągu znaków.

Zmienna wyliczeniowa

- **Wyliczenie** jest listą wartości stałych całkowitych, np. :

```
enum boolean { NO, YES }
```

czyli po słowie **enum** podajemy nazwę stałej wyliczeniowej, a w nawiasach *{ }* nazwy wartości, które może przyjmować.

  - Jeżeli nie określimy inaczej, nazwy mają wartości liczone od zera z krokiem co 1, czyli *NO* – 0, *YES* – 1.
  - Inny przykład:

```
enum rok { JAN = 1, FEB, MAR, APR, MAI, JUN, JUL, AUG, SEP, OCT, NOV, DEC }
```

    czyli *JAN* ma wartość 1, a wtedy kolejno *FEB* - 2, *MAR* - 3, itd. aż do *DEC* – 12. Gdybyśmy *JAN* przypisali 2, wtedy *DEC* miałby wartość 13, itp.

- Nazwy występujące w różnych stałych wyliczeniowych nie mogą się powtarzać. W tym samym wyliczeniu mogą jednak powtarzać się wartości liczbowe przypisane do unikalnych nazw (choć trzeba je wtedy zazwyczaj ręcznie ustawiać).

Typ wyliczeniowy

- **Typ wyliczeniowy** – jest to typ, który służy do tworzenia zmiennych, które powinny przechowywać tylko pewne, z góry ustalone wartości.
- Definicja:

```
enum  id_typu   {   lista_stałych   } id_zmiennej;
```

- Przykłady:

```
enum dni {ni, po, wt, sr, cz, pi, so};
        /* ni == 0, po == 1, ... , so == 6 */
enum dni {ni=1, po, wt, sr, cz, pi, so};
        /* ni == 1, po == 2, ... , so == 7 */
dni Egzamin, Dobry = cz;
Egzamin = Dobry;
     //wartości w tym samym wyliczeniu mogą się powtarzać
enum TW1 {t1, t2, t3 = 0, t4, t5, t6 = 1, t7};
        /* t2 == 1 t4 == 1, t5 == 2, t7 == 2 */

enum {A = 0x41, B, C, X = 0x58} znak;
                // zmienna typu wyliczeniowego
znak = C;                   // poprawnie
//można przypisywać pod typ wyliczeniowy liczby, nawet nie mające
// odpowiednika w wartościach a kompilator może o tym nie ostrzec
znak = 0x49;      // błąd
znak = 0x41;      // błąd – nie ma konwersji z int do znak; znak = A; - ok
```

Typ wyliczeniowy - Przykład

```
#include <stdlib.h>       #include <time.h>         #include <conio.h>
enum Dziedzina { ASTRONOMIA, MATEMATYKA, FIZYKA, CHEMIA, BIOLOGIA };
int main(int argc, char* argv[]){
    enum Dziedzina pytanie;
    srand((int)time(0));
    pytanie = (enum Dziedzina)(rand() % 5);    /* losowanie dziedziny */
    switch (pytanie){
        case ASTRONOMIA:
           printf("Ksiezyca w nowiu nie widac - wiec gdzie on jest ? ");
           break;
        case MATEMATYKA:
           printf("Ktorych liczb jest wiecej: calkowitych czy rzeczywistych ?");
           break;
        case FIZYKA:
           printf("Co przyciaga silniej - Slonce czy Ksiezyc ? ");
           break;
        case CHEMIA:
           printf("Dlaczego uzywamy moli a nie gramów ? ");
           break;
        case BIOLOGIA:
           printf("Czym sie rozni wrobel od mazurka ? ");
           break;
    }
    printf("\n\n");
    getch();
    return 0;
}
```

Zmienne

- Zmienne przed użyciem muszą być zadeklarowane, tj. należy określić ich typ, a następnie wymienić jedną lub kilka zmiennych tego typu, np.:

```
int a, b, c;   // przykład deklaracji
int a;         // to samo co wyżej
int b;         // ale pozwala
int c;         // skomentować każdą zmienną
```

- Wartości początkowe mogą być nadane zmiennym już na poziomie ich deklaracji (po nazwie zmiennej występuje znak = i pewne wyrażenie pełniące funkcję inicjatora):

```
int a = 10;
int b = 60;
int c = a;     //czyli 10
float eps = 1.0e-5;
```

- **Zmienne zewnętrzne (globalne) i statyczne** przez domniemanie mają nadaną wartość początkową zero.
- **Zmienne automatyczne (lokalne)** bez jawnie określonej wartości początkowej mają wartości przypadkowe.

Zmienne

- **Typy danych dostępne w języku C przy deklaracji zmiennych i zależności między nimi:**

```
char            signed char
int             signed              signed int
short           short int           signed short int
long            signed long         long int            signed long int
long long       signed long long
unsigned char
unsigned int     unsigned
unsigned short   unsigned short int
unsigned long    unsigned long int
unsigned long long
float
double
long double
```

Zmienne

- **Przykładowe deklaracje/definicje zmiennych**
  - Deklaracja informuje jedynie kompilator czym jest tworzony obiekt, ale nie rezerwuje dla niego miejsca w pamięci. Stąd też deklarować zmienną w programie można wielokrotnie (np. **extern int** *liczba*).
  - Definicja dodatkowo rezerwuje pamięć dla tworzonego obiektu. Każda definicja jest też deklaracją.

```
int   i;
char   a, b, c;
unsigned long    duza_odleglosc;
float KursDolara;
double   masa, gestosc;
int   licznik = 125,   suma = 0;
float   dokladnosc = 0.0005, uchyb = 0.001;
double   moc = 15e6,   straty = 1500;
double alfa = 3.34, beta, jota = 15.15, kappa;
```

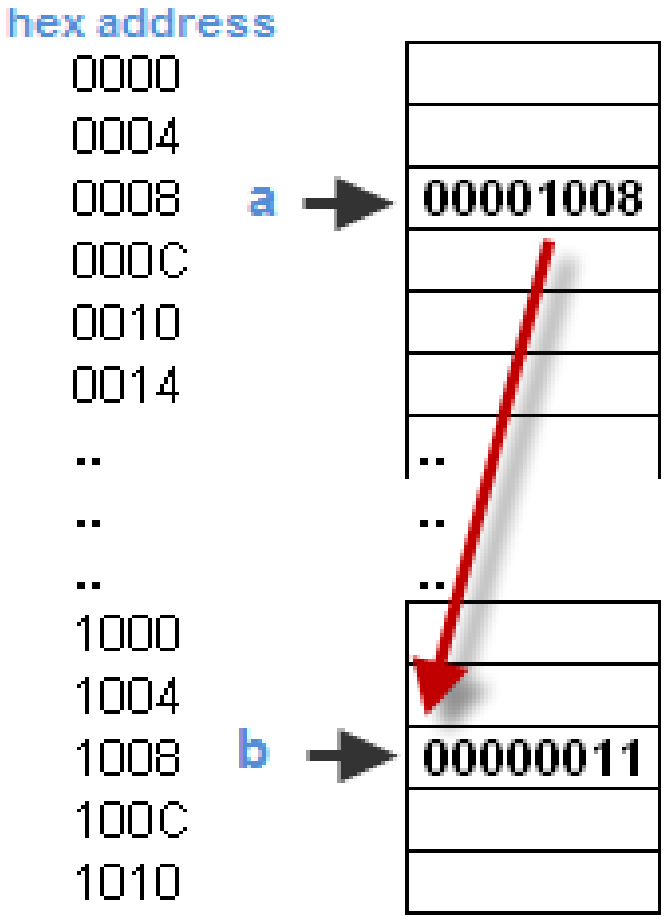
Notacja węgierska

- Nie istnieje jedna konwencja nazewnicza:
  - oddzielanie twardą spacją: int_array_size
  - konwencja "pascalowska": IntArraySize
  - konwencja "wielbłądzia": intArraySize
- **Notacja węgierska** jest to sposób zapisu nazw zmiennych oraz obiektów, polegający na poprzedzaniu nazwy małą literą (literami) określającą rodzaj tej zmiennej (obiektu).
- Przykład użycia notacji węgierskiej do nazywania zmiennych w języku C:

Przedrostek	Typ danych	Przykład
b	bool	`bJeszczeRaz`
c	char	`cKodPolecenia`
l	long	`lDuzyKaliber`
n	int	`nLicznikPierwszy`
p	wskaźnik	`pAdresNowejCeny`
a	tablica	`anDaneTestowe`
s	łańcuch znaków	`sStosownyNapis`

Wskaźniki

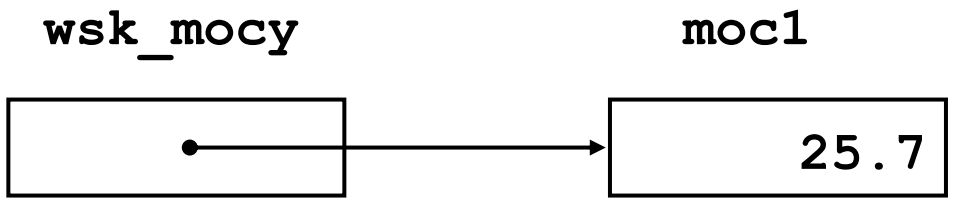
- Wskaźnik, jak sugeruje sama nazwa, COŚ wskazuje. Tym czymś jest adres pamięci operacyjnej komputera, pod którym umieszczona jest jakaś zmienna czy np. element tablicy
- Pamięć, w typowej maszynie, jest to tablica kolejno numerowanych lub adresowanych komórek pamięci (można nimi manipulować indywidualnie albo całymi grupami sąsiadujących komórek).
- Wskaźnik to grupa komórek (często 2 lub 4), które mogą pomieścić adres.
- Na rysunku ***a*** to wskaźnik (gdzieś w pamięci), wskazujący na KONKRETNE położenie zmiennej ***b***.
- Wskaźnik zazwyczaj zajmuje 4 bajty (bez względu na jaki typ danych wskazuje).



Zmienne wskaźnikowe: deklaracje

- **Przykłady:**

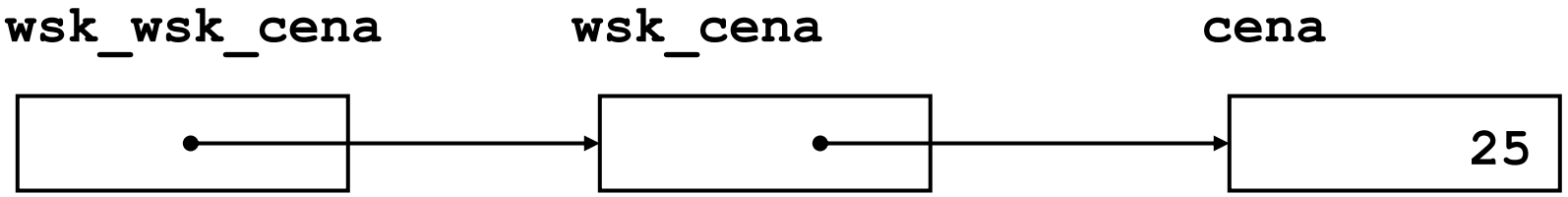
```
int    *pt_i,  *pt_j;
double   *wsk1,  *wsk2;
float   moc1 = 25.7,  moc2,  *wsk_mocy = &moc1;
void   *dowolny,  *kazdy;
```



Zmienne wskaźnikowe: deklaracje

- **Przykłady:**

```
int cena = 25, *wsk_cena, **wsk_wsk_cena;
wsk_cena = &cena;
wsk_wsk_cena = &wsk_cena;
```




```
int    i = 5,    j = 7;
int    *pt = & i,  *pk = & j;
double droga, czas = 100, *wsk1,
       *wsk_param = & droga;
```

Operator wyznaczania wskaźnika &

- Jednoargumentowy operator : **&** podaje adres obiektu.
- Na przykład polecenie

```
p = &c; // int *p, c;
```

przypisuje wskaźnikowi *p* pewien adres w pamięci, w którym znajduje się zmienna *c*. Innymi słowy, od teraz zarówno *p* (wskaźnik) jak *c* (nazwa zmiennej) wskazują na to samo (i mogą to coś modyfikować)

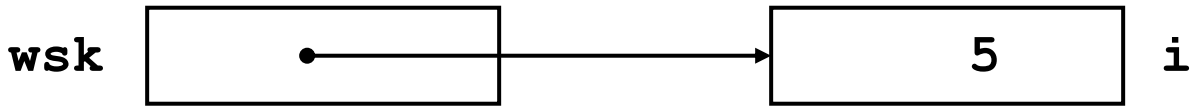
- Przykłady:

```
int lampy, widelce;
int *wsk_towaru;
wsk_towaru = & lampy;
.................
wsk_towaru = & widelce;

float   Korzysc = 2.54, *wsk_f;
long    *wsk_l;
void    *wsk_v;

wsk_f = & Korzysc;          // poprawnie
wsk_l = & Korzysc;          // błąd, nieprawidłowy typ (long vs float)
wsk_v = & Korzysc;    // poprawnie, ~~void ->long~~  void -> float
                      // ale gdyby próbować odczytać to:
                      // float new_f = *((float*)wsk_v);
```

Operator dostępu pośredniego *

- Operator: * (gwiazdka, operator *adresowania pośredniego*, albo po prostu: *operator wskaźnikowy*).
- Zastosowany do wskaźnika daje **zawartość obiektu** wskazywanego przez ten wskaźnik.
- **Przykłady:**

```
int i = 5, j;
int *wsk = & i;
j = *wsk;              // równoważne j = i;
```




```
int x = 1;
int y = 2;
int *ip;                    // deklaracja wskaźnika do wartości int
ip = &x;                    // ip wskazuje na to co zmienna x, czyli na 1
y = *ip;                    // teraz y ma wartość 1, równoważne y = x;
*ip = 0;                    // teraz zmienna x ma wartość 0, równoważne x = 0;
```

Zmienne wskaźnikowe

- W momencie gdy używamy nazwy zmiennej, odwołujemy się do wartości na którą ta zmienna wskazuje:

```
int x = 0; // przypisz 0 jako wartość x
```

- Gdy używamy samej nazwy wskaźnika (bez *), odwołujemy się do adresu w pamięci RAM. W związku z tym przypisanie *bezpośrednio* do wskaźnika jakiejś wartości nie ma sensu (**bezpośrednio**, czyli BEZ *):

```
int *p;         // p - wskaźnik na typ int
p = 10;         // błąd - pod 'p' kryje się adres RAM (bo jest to wskaźnik
                // którego to adresu nie da się przewidzieć pisząc kod
```

- Widać więc, że bez użycia dodatkowych operatorów (* i &), wartości numerycznej i wskaźnika nie można (bezpośrednio) przypisywać do siebie (operatorem =):
  - & - pobiera adres zmiennej
  - * przy wskaźniku wskazuje na WARTOŚĆ, jaka jest pod jego (wskaźnika) adresem.

Zmienne wskaźnikowe

```
int *p;           // wskaźnik na typ int
int x = 15;       // zmienna x o wartości 15
p = &x;           // przypisuje wskaźnikowi p ADRES zmiennej x

p = 5546;         // BŁĄD, nie można bezpośrednio przypisać adresu
p = 0xFA744EA4;   // BŁĄD jak wyżej, to, że wygląda jak prawidłowy adres 32bitowy
                  // nie znaczy, że programista go zna w momencie pisania kodu....
                  // poniższe dwa polecenia robią dokładnie to samo na zmiennej x:

*p = 30;          // przypisz zmiennej x wartość 30
                  // używając 'JEJ' wskaźnika (bo wcześniej: p = &x; )
x = 30;           // przypisz zmiennej x wartość 30 BEZPOŚREDNIO
int y = 0;        // nowa zmienna

                  // poniższe trzy polecenia też robią dokładnie to samo:

y = *p;           // przypisz do y wartość na jaką wskazuje wskaźnik p
                  // czyli wartość zmiennej x (czyli 30)
y = x;            // przypisz wartość zmiennej x do zmiennej y BEZPOŚREDNIO
y = 30;           // przypisz wartość 30 zmiennej x bezpośrednio
```

Zmienne wskaźnikowe

- Załóżmy następujący przykład: wskaźnik ip wskazuje na zmienną x:

```
int x = 10;
int *ip;
ip = &x;  // przypisz do ip adres zmiennej x, czyli
          // "ip wskazuje na x",
          // alternatywnie: int *ip = &x; w jednej linii
```

- Od teraz wszędzie tam, gdzie jest potrzeba użycia zmiennej x bezpośrednio, ***można*** użyć jej wskaźnika z operatorem * (dlatego właśnie operator * nazywamy *operatorem adresowania pośredniego*), np.

```
// poniższe dwa polecenia robią to samo, ponieważ
// ip oraz x odnoszą się do tej samej wartości int w RAM
*ip = *ip + 10;
x = x + 10;
```

Referencje

- Referencja jest jakby przezwiskiem jakiejś zmiennej. Dzięki referencji na tą samą zmienną można mówić używając jej drugiej nazwy.
- Każda operacja wykonywana na referencji jest identyczna z operacją wykonaną bezpośrednio na reprezentowanej przez tą referencję zmiennej czy strukturze danych.

- **Przykłady:**

```
int kwota;
int &ref_k = kwota;       // adres do referencji można przypisać tylko raz
ref_k = 1254;             // równoważne kwota = 1254;

long a, b, &ref_a = a;
ref_a = 12;               // równoważne a = 12;
b = ref_a;                // równoważne b = a;

float moc_x, &ref_x = moc_x, *wsk_x;
wsk_x = & ref_x;          // równoważne wsk_x = & moc_x;
wsk_x = ref_x;            // błąd, podobnie jak wsk_x = moc_x;
```

Stałe – rozkaz *const*

- Kwalifikator **const** można stosować do deklaracji dowolnej zmiennej.

- Zmienna zadeklarowana z takim słowem (przed nazwą typu) informuje kompilator, że jej raz nadanej wartości (bezpośrednio w deklaracji) nie można zmieniać. Taka próba kończy się najczęściej błędem, i to już na etapie kompilacji (zależnie od kompilatora). Np.:

```
const float pi = 3.14;
const double e = 2.71828182845905;
```

- Można używać **const** w deklaracjach tablic (żaden element takiej tablicy nie może zostać zmieniony), czy też w tablicowych parametrach funkcji (funkcja nie ma prawa zmieniać nic w takiej tablicy). Np.:

```
const char msg[] = "Uwaga:  ";
int funkcja(const int[]);
```

- Stałe podobnie jak zmienne można wymieniać po przecinku. Np.:

```
const int dni = 7, tygodnie = 52;
const float pi = 3.14159, e = 2.71828;
const double Avogadro = 6.022E23;
```

Pytania?