Podstawy programowania.

Wykład 3

Zmienne i operatory

Typy danych

- Podstawowe wbudowane typy danych języka C:
  - _Bool – 0 i 1 (C99)
  - znaki (*char*) – 7 bitów dla znaków ASCII
  - liczby całkowite (*int*), od INT_MIN do INT_MAX (z <limits.h>)
  - liczby rzeczywiste (zmiennoprzecinkowe)
    - pojedynczej (*float*), podwójnej (*double*) i rozszerzonej (*long double*) precyzji (standardy IEEE)
    - liczby zespolone: *float _Complex*, *double _Complex* i *long double _Complex* (C99 - opcjonalne)
- określenia (specyfikatory): *short, long, signed, unsigned*
  - sugerują jak wiele miejsca w pamięci przeznaczyć na zmienną i czy przeznaczyć jeden z bitów na przechowywanie znaku zmiennej
    - *signed char, short int, int, long int, long long int* – zmienne całkowite ze znakiem
    - dla każdego typu ze znakiem istnieje odpowiednik bez znaku - *unsigned*

Stałe (literały)

- Stałe typu całkowitego (oprócz standardowych):
  - long – np. `2L`
  - unsigned – np. `12u`
  - unsigned long – np. `25UL`
- Zapis stałych typu całkowitego
  - standardowy – jak wyżej
  - ósemkowy (*octal*) – `037 == 31`
  - szesnastkowy (*hexadecimal*) – `0x1F == 31`
  - `0` na początku zawsze oznacza notację ósemkową, a `0x` notację szesnastkową (możliwe warianty *long*, *unsigned*) – `0xFUL == 15UL`
- Stałe typu znakowego
  - małe liczby całkowite, zapisane w specyficznej notacji – `'x'`
    - `'0' == 48`
    - *escape sequences* – np. `'\b'`, `'\n'` `'\t'`, `'\\'`, `'\?'`, `'\"'`, `'\''`
      - zapis ósemkowy: `'\170'`, zapis szesnastkowy: `'\x5D'`

Stałe (literały)

- Stałe napisowe
  - "Hello world\n"
  - tablice znaków zakończone znakiem NULL - '\0'
  - długość napisu (np. zwracana przez funkcję *strlen*) zazwyczaj nie obejmuje '\0' – alokacja pamięci do przechowywania napisu musi to uwzględniać
  - "x" != 'x'
- Stałe typu zmiennoprzecinkowego
  - float – 3.14f
  - double – 3.14
  - long double – 3.14L
- Wyrażenia stałe
  - wyrażenia zbudowane wyłącznie ze stałych – 3.14+1
  - wyrażenia stałe zazwyczaj obliczane są w trakcie kompilacji

Programowanie

- Dane:
  - stałe (z nazwą lub bez)
  - zmienne (na razie tylko z nazwą)
- Nazwy zmiennych (i stałych)
  - mogą zawierać litery, cyfry, znak podkreślenia _
  - nie mogą być słowami kluczowymi języka
  - nie mogą zaczynać się od cyfry
  - duże litery są różne od małych
  - powinny ułatwiać zrozumienie programu:
    - np. rok_urodzenia, SredniaOcena itp.
  - konwencje:
    - od znaków _ często zaczynają się nazwy zmiennych w bibliotekach
    - stałe często mają nazwy z samych dużych liter
    - inne: np. rozróżnienie argumentów i zmiennych lokalnych

Nazwy zmiennych i słowa kluczowe C

- słowa kluczowe (*keywords*):

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

+ słowa zaczynające się od _ (np. _Bool)

- nazwy w aplikacjach powinny stosować własne konwencje
  - nazwy mogą składać się z różnych składników, z których np. pierwszy określa program:
    - mpp_.... (mój pierwszy program itd. np., mpp_liczba_obiektów)

Formatowane wejście/wyjście

- Podstawy stosowania funkcji printf:
  - pierwszy argument określa sposób formatowania
  - kolejne argumenty podają dane do wydruku
    - mogą to być zmienne lub stałe
  - format to sposób konstruowania napisu
    - zadany bezpośrednio – napis (łańcuch znaków)
      - znaki ASCII, znaki sterujące (\t, \n), znaki UTF (uwaga!)
      - jak w programie "hello"
    - symbolicznie definiujący reprezentację zmiennych typów liczbowych
      - %d, %f, %lf, %c, %s
      - %x.ylf – x rozmiar napisu, y – liczba miejsc po przecinku (np. %20.12lf)

Formatowane wejście/wyjście

- Podstawy użycia funkcji *scanf*
  - argumenty podobne jak dla funkcji *printf*
    - napis będący wzorcem formatowania
    - lista argumentów pasujących do symboli określających wstawienie zmiennej określonego typu
      - %d, %f, %lf, %c, %s i inne – podobnie jak dla printf
    - **argumentami na liście są zawsze adresy, miejsca wstawienia odczytywanych wartości zmiennych**
  - scanf posiada precyzyjne zasady wczytywania wartości zmiennych zgodnie z wzorcem
    - znaki odstępu, tabulacji, nowej linii są pomijane
  - **najprościej, najwygodniej i najbezpieczniej jest korzystać ze scanf wyłącznie do odczytania jednej wartości zmiennej zgodnie z najprostszym wzorcem:**
    - **scanf("%d", &i); scanf("%lf", &a);**

Instrukcje i operatory

- Kod źródłowy zawiera instrukcje
- Instrukcje mogą być wyrażeniami złożonymi z:
  - stałych (1, 2.0, 'a', "napis", 037 (==31), 0x1f (==31))
  - nazw – zmiennych i stałych, ale także funkcji
  - operatorów
- Operator
  - związany jest z jednym lub dwoma, rzadko z trzema argumentami
  - produkuje wynik
    - dla argumentów określonego typu wynik posiada ściśle określony typ
    - zasady użycia operatorów określają możliwe typy argumentów i wyniku
      - definicje typów określają możliwość stosowania konkretnych operatorów

Operatory

- Podstawowe operatory:
  - przypisanie (też jest operatorem i zwraca wynik...):
    - typ – lewy argument
    - wartość – po przypisaniu
      - a=1; b=a+2;
    - wyrażenie po lewej stronie (*lvalue*) – obszar pamięci
  - jednoargumentowe operatory arytmetyczne: +, -, ++, --
    - -3, -a, c++, --k
    - a = c++; // wartość a różna niż w przypadku a = ++c: !
  - operatory arytmetyczne: +, -, *, /, %
    - a+b, c – h*k, 4*n, d/1.0; // zasady priorytetów i łączności
    - k/4, m%7; // % tylko dla liczb całkowitych
  - złożone operatory przypisania: +=, *=, -=, /=, %=
    - » a op= b // równoważne a = a op b
    - » także dla operatorów bitowych <<=, >>=, &=, ^=, |=

Operatory

- Podstawowe operatory:
  - operatory relacji: <, >, <=, >=, ==, !=
    - wynik jako liczba całkowita – 0 lub 1
    - a<b, c >= d, f==10, wartosc_logiczna != false
  - operatory logiczne: &&, || (wynik jako liczba całkowita)
    - a<b && c>=d, d==7 || d <= 4 // kolejność obliczania od lewej
    - if(a>b || c<d) {...}; // jeśli pierwszy warunek prawdziwy, drugi nie jest obliczany
    - if(a>b && c<d) {...}; // jeśli pierwszy warunek fałszywy, drugi nie jest obliczany
  - jednoargumentowe operatory logiczne: ! (negacja)
    - zamienia wartości różne od zera w zero, a zero w jeden
    - if(!wartosc_logiczna) // równoważne if(wartosc_logiczna==0)

Operatory

- Trzyargumentowy operator "?:" stosowany w wyrażeniach:

  `w_1 ? w_2 : w_3` jest interpretowany następująco:
  - jeżeli wyrażenie `w_1` jest prawdziwe, obliczane jest wyrażenie `w_2` i ono stanowi wynik działania operatora
  - jeżeli wyrażenie `w_1` nie jest prawdziwe, obliczane jest wyrażenie `w_3` i ono stanowi wynik działania operatora
  - tylko jedno z wyrażeń `w_2` i `w_3` jest obliczane
  - typ wyniku wynika z zasad konwersji dla operatorów dwuargumentowych (w tym wypadku dotyczy `w_2` i `w_3`)
  - przykład:
    - `printf("You have %d item%s.\n", n, n==1 ? "" : "s");`
- Wyrażenie:

  `c = a > b ? a : b`

  jest równoważne sekwencji operacji obliczania maksimum:

  `if (a>b) c=a; else c=b;`

Operatory

- Zasady języka określają
  - priorytety operatorów (siłę wiązania z argumentami)
  - kierunek odczytywania argumentów
- Kolejność wykonywania operacji w wyrażeniu wynika z:
  - priorytetów operatorów
  - użycia nawiasów

Operators	***Read from***
Unary operators: ! ~ ++ -- + - * & (*typecast*) sizeof	Left to right
* / %	Right to left
+ –	Left to right
<< >	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
^	Left to right
|	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= *= /= %= &= ^= |= <<= >=	Right to left
,	Left to right

Operatory

- Konwersje typów:
  - jawna konwersja ma postać: *(nowy_typ) wartość:*
    - `int a = (int) 3.5; float f = (float) d;` // double d = 0.1;
  - istnieje możliwość dokonywania przez kompilator niejawnej konwersji przy przypisaniu (a także dla innych operatorów)
    - `int a = 3.5; float f = 0.1;` // 0.1 - double, 0.1f - float
  - dla operatorów dwuargumentowych niejawna konwersja sprowadza oba do wspólnego typu, tak aby nie utracić precyzji
  - zapis kodu może oznaczać konieczność konwersji tracącej dokładność
    - kompilator może zgłosić ostrzeżenie
    - taka konwersja nie stanowi błędu składni
  - **konwersja może oznaczać utratę dokładności (zaokrąglenia), czasem może także prowadzić do zupełnie błędnych wyników**
  - **niejawne konwersje mogą zmniejszać czytelność kodu, ukrywać intencje programisty**

Podstawowe typy zmiennych

```
#include <stdio.h> // USES
int main(void)  // zwrot kodu błędu lub sukcesu (0)
{
   // definicje (deklaracje – informacje o nazwie i typie + rezerwacja
   int n;   char c;   //                                    obszaru w pamięci)
   // inicjowanie
   n = 1/3; // instrukcja przypisania
   c = 'a';
   // definicja z inicjowaniem
   float f = 1.0/3.0;
   double d = 1.0/3.0; // zapis
   // d = f; efekt konwersji?  (co w przypadku f = d ?)
   // możliwe notacje dla liczb zmiennoprzecinkowych
   printf("liczby i znaki: %c, %7d, %20.15f, %.15lf\n", c, n, f, d);
   return(0);  // obsługa błędów - temat rzeka
}
```

Zapis binarny

- *Zapis liczb rzeczywistych*
  - formaty zmiennoprzecinkowe IEEE 754
  - pojedyncza precyzja, SP
    - 32 bity (znak + 23 bity mantysy + 8 bitów wykładnika)
  - podwójna precyzja, DP
    - 64 bity (znak + 52 bity mantysy + 11 bitów wykładnika)
  - mantysa znormalizowana (brak zapisu bitu przed przecinkiem)
    - zakres dla liczb dodatnich:
      - SP: od 1.2×10^{-38} do 3.4×10^{38}, DP: od 2.2×10^{-308} do 1.8×10^{308}
    - dokładność w cyfrach znaczących: SP – 6 do 9, DP – 15 do 17
  - operacje na liczbach zmiennoprzecinkowych IEEE 754
    - dodatkowa dokładność w trakcie realizacji operacji
    - pułapki i wyjątki (nadmiary, niedomiary, błędy operacji)
    - wartości specjalne (+/-0, +/-nieskończoność, NaN)

Zapis binarny

- *Zapis liczb rzeczywistych – specyfika i problemy*

  (wydruki z 20 cyframi znaczącymi)
  - float f = 1.0/3.0; // (f = 0.33333343267440796)
    - tylko 7 pierwszych cyfr jest znaczących
  - double d = 1.0/3.0; // (d = 0.33333333333333315)
    - 16 cyfr znaczących
  - operacje na liczbach zmiennoprzecinkowych mogą dodatkowo zmniejszać liczbę cyfr znaczących (z powodu konieczności przekształcenia do tego samego wyznacznika przed operacją)
    - matematycznie: 1/0.0000123 = 81300,(81300)
    - double d = 1.0/0.0000123; (d = 81300.8130081300769)
      - 15 cyfr znaczących
    - double d = 1.0/((123000+0.0000123)-123000);
      - matematycznie identyczne z 1/0.0000123
      - w rzeczywistości (d = 81300.771057083708)
      - zostało tylko 5 cyfr znaczących!

Operacje na zmiennych

- *Problemy dla reprezentacji z ograniczoną dokładnością*
  - problemy z dokładnością: ile bitów potrzeba żeby dokładnie zapisać liczbę 0.1?
  - problemy z arytmetyką
    - $x * (1/x) \neq 1$ – dla 135 spośród pierwszego tysiąca liczb naturalnych przy zapisie z pojedynczą precyzją
    - $a + b = a$ (dla b>0 ale znacznie mniejszego od a)
      - co z 1/((a+b)-a) ?
    - (a+b)-a != (a-a)+b - brak łączności i przemienności operacji
    - przydatność poprawy dokładności w trakcie dokonywania obliczeń
    - rozwiązanie niektórych problemów za pomocą zmiany algorytmu