Język C i C++. Podstawy

Materiały do samodzielnego opanowania, ale także propozycja zadań na zajęcia laboratoryjne.

Zagadnienia do opanowania

1) Jak działa debuger? Co oznacza określenie „śledzenie krokowe”?
2) W jaki sposób można śledzić krokowo aplikację? Jak można szybko (bez śledzenia) wykonać wszystkie wcześniejsze instrukcje, zatrzymując się w konkretnym miejscu kodu?
3) Co to jest rzutowanie typów? Czym różni się rzutowanie jawne od niejawnego?
4) Jaką instrukcję wykonuje procesor, gdy użyjemy dzielenia **`2/3`**, a jaką, gdy **`2.0/3.0`**?
5) Jaki będzie typ wyniku działań **`2/3.0`** oraz **`2.0/3`**?
6) Jakie są ograniczenia typów zmiennoprzecinkowych pojedynczej i podwójnej precyzji?
7) Jakie skutki spowoduje użycie argumentów prinft(…) o typach niezgodnych z odpowiadającymi im ciągami formatującymi, np. **`printf("%d", 2.5);`** albo **`printf("%g", "Ala ma kota");`** Zwróć szczególną uwagę na przypadki takie jak **`printf("%d %d %d", 1.1, 2.2, 3.3);`**
8) Napisz aplikację (używając 2 pętli for), aby wyświetlić wyniki poniższych działań. Wyniki wyświetlaj jako **`%d`**, **`%g`** albo **`%.5f`** w zależności od potrzeb. Symbole **`a`** oraz **`b`** to liczby całkowite (int) z zakresu od 1 do 5 włącznie, co daje nam 25 wierszy kombinacji (kilka z nich przedstawiono poniżej). W jaki sposób są rzutowane typy int-double? Jaki to ma wpływ na część ułamkową? Jak uzyskać zaokrąglenie z nadmiarem? A jak z niedomiarem?

```
| a | b |a*b |a/b|a%b |(double)a/(double)b|
=========================================
| 1 | 1 |  1 | 1 |  0 |      1.00000      |
| 1 | 2 |  2 | 0 |  1 |      0.50000      |
...
| 5 | 4 | 20 | 1 |  1 |      1.25000      |
| 5 | 5 | 25 | 1 |  0 |      1.00000      |
```

9) Przyjrzyj się wynikom w kompletnej tabelce z punktu 8. Jak działa operator %? Co jest wynikiem działania a % b ?
10) Spośród liczb z zakresu 1..1000 wyświetl wszystkie podzielne przez 17.
11) Spośród liczb z zakresu 1..10000 wyświetl wszystkie nieparzyste liczby podzielne przez 123.

Rozmyty obrazek to jedno z możliwych rozwiązań zadania 11 wraz z wynikami. Kod bez trudu mieści się w 4 liniach ładnie sformatowanego tekstu, spostrzegawczy programista jest w stanie uprościć go do 2-3 linijek (i nie chodzi tu oczywiście o wykasowanie znaków końca linii).

Przykład 1. (kod do samodzielnej modyfikacji)

```
#include <stdio.h>
int main()
{
     printf("%d\n", 1/3);       printf("%f\n", 1/3);
     printf("%d\n", 1.0/3.0);   printf("%f\n", 1.0/3.0);
     return 0;
}
```

Przykład 2.

```
#include <stdio.h>
int main() //możesz użyć debugera, aby obejrzeć wartości zmiennych
{
     int i = 2/3;
     double d0 = 2/3;
     double d1 = 2.0/3;
     int j = 2.0/3.0;
     return 0;
}
```

Przykład 3.

```
#include <stdio.h>
int main()
{
    //przypadek A
     printf("A: %d %d %d %d\n", 1.0/3.0, 1/3, 1.0/3, 1/3.0);

    //przypadek B
    int a = 1.0/3.0, b = 1/3, c = 1.0/3, d = 1/3.0;
    printf("B: %d %d %d %d\n", a, b, c, d);

    return 0;
}
```

Powyższy przykład pozornie powinien dać identyczne rezultaty, ale kiedy obejrzymy wyniki, to widzimy:

```
A: 1431655765 1070945621 0 1431655765
B: 0 0 0 0
```

Dlaczego tak jest? (wskazówka: kiedy dokładnie ma miejsce niejawna konwersja typów?)

Spróbuj poprawić kod „A" w taki sposób, aby uzyskać dobre wyniki (tzn. takie jak w „B"). Zrób to na dwa sposoby – zmieniając ciągi formatujące, oraz posługując się jawnym rzutowaniem typów.

Przykład 4.

```
#include <stdio.h>
int main()
{
     int n, ile;
     do {
          printf("Wprowadz dodatnia liczbe calkowita: ");
          fflush(stdin);
          ile = scanf("%d", &n);
     } while (ile != 1 || n <= 0);
     if (n % 2 == 0)
          printf("Liczba %d jest parzysta.\n", n);
     else
```

```
        printf("Liczba %d jest nieparzysta.\n", n);


    return 0;
}
```

Symbol || (dwie pionowe kreski) to operator OR (lub) używany w warunkach. Wyrażenie **`do {…} while (ile != 1 || n <= 0)`** oznacza: wykonuj … tak długo, jak długo zmienna ile ma wartość różną od 1 lub wprowadzona liczba n nie jest dodatnia.

Niedługo na wykładzie pojawi się nowa instrukcja, która pozwoli kilka ostatnich linijek uprościć do zwięzłej (i niekoniecznie czytelnej) postaci:

```
printf("Liczba %d jest %sparzysta.\n", n, n%2 ? "nie" : "");
```