

Wydział budowy maszyn i informatyki

Architektura komputerów

(Laboratorium №1)

Temat ćwiczenia: Systemy kodowania liczb

Data wykonania ćwiczenia: 11.10.2023

Igor Gawłowicz

Wiktoria Mrózek

Cel ćwiczenia:

Celem ćwiczeń z systemami kodowania liczb jest wprowadzenie w zagadnienia reprezentacji liczb w różnych systemach liczbowych, zrozumienie precyzji obliczeń, oraz rozwijanie umiejętności analitycznych, co jest istotne w dziedzinach związanych z informatyką, elektroniką, programowaniem i inżynierią.

Przebieg ćwiczenia:

1. W oparciu o poniższy program, wyświetl zawartość pamięci dla różnych typów danych i porównaj binarne reprezentacje danych, w szczególności sprawdź typy int, float oraz double, definiując parametry o różnych wartościach. Dodatkowo, zbadaj zawartość pamięci dla maksymalnych i minimalnych wartości dla różnych typów. W celu wykonania modyfikacji, należy zmieniać odpowiednie wartości zmiennych w funkcji głównej (main). Poniżej znajduje się kod programu:

```
int main() {
    int a = 42;
    show_32_bits(a);

    float b = 3.14;
    show_32_bits(b);

    double c = 2.71828;
    show_64_bits(c);

    int d = std::numeric_limits<int>::min();
    show_32_bits(d);

    int e = std::numeric_limits<int>::max();
    show_32_bits(e);

    double f = std::numeric_limits<double>::min();
    show_64_bits(f);

    double g = std::numeric_limits<double>::max();
    show_64_bits(g);

    return 0;
}
```


2. 3. Napisz program lub funkcję w języku C++ do sumowania dwóch liczb zmiennoprzecinkowych i porównania wyniku z oczekiwaną wartością.

```
bool porownajSume(double liczba1, double liczba2, double oczekiwanaWartosc) {
    double suma = liczba1 + liczba2;
    return suma == oczekiwanaWartosc;
}

int main() {
    double liczba1, liczba2, oczekiwanaWartosc;

    // Wprowadź dwie liczby zmiennoprzecinkowe
    std::cout << "Podaj pierwsza liczbe: ";
    std::cin >> liczba1;

    std::cout << "Podaj druga liczbe: ";
    std::cin >> liczba2;

    // Wprowadź oczekiwaną wartość
    std::cout << "Podaj oczekiwana wartosc sumy: ";
    std::cin >> oczekiwanaWartosc;

    // Wywołaj funkcję porównującą sumę
    bool wynik = porownajSume(liczba1, liczba2, oczekiwanaWartosc);

    if (wynik) {
        std::cout << "Wynik jest zgodny z oczekiwaną wartoscia." << std::endl;
    } else {
        std::cout << "Wynik nie jest zgodny z oczekiwaną wartoscia." << std::endl;
    }

    return 0;
}
```

Podaj pierwsza liczbe: 6.3
Podaj druga liczbe: 1.7
Podaj oczekiwana wartosc sumy: 8.0
Wynik jest zgodny z oczekiwaną wartoscia.

Podaj pierwsza liczbe: 19.4
Podaj druga liczbe: 0.5
Podaj oczekiwana wartosc sumy: 19.9
Wynik jest zgodny z oczekiwaną wartoscia.

Podaj pierwsza liczbe: 117.11
Podaj druga liczbe: 298.546
Podaj oczekiwana wartosc sumy: 415.656
Wynik jest zgodny z oczekiwaną wartoscia.

2. 4. Napisz program lub funkcję w języku C++, która zawiera pętlę, która 100 razy dodaje wartość 0.1 do zmiennej "suma". Celem zadania jest zrozumienie, dlaczego wynik obliczeń może być inny niż oczekiwany ze względu na niedokładność reprezentacji liczb zmiennoprzecinkowych. Wykorzystaj funkcje biblioteki do bardziej dokładnego obliczenia wartości oczekiwanej i porównania jej z wynikiem rzeczywistym. Możesz użyć funkcji `std::abs()` do obliczenia wartości bezwzględnej różnicy między wynikiem rzeczywistym a oczekiwanym. Porównaj różnice między wartościami rzeczywistymi a oczekiwanymi w różnych iteracjach pętli. Zauważ, czy błąd reprezentacji liczb zmiennoprzecinkowych narasta z każdą iteracją.

```
int main() {
    double suma = 0.0;
    double oczekiwanaWartosc = 0.1 * 100; // Wartość oczekiwana: 0.1 * 100

    for (int i = 0; i < 100; i++) {
        suma += 0.1;
    }

    double roznica = std::abs(suma - oczekiwanaWartosc);

    std::cout << "Wynik rzeczywisty: " << suma << std::endl;
    std::cout << "Wartość oczekiwana: " << oczekiwanaWartosc << std::endl;
    std::cout << "Różnica: " << roznica << std::endl;

    return 0;
}
```

```
Wynik rzeczywisty: 10
Wartość oczekiwana: 10
Różnica: 1.95399e-14
```

Niedokładność reprezentacji liczb zmiennoprzecinkowych w komputerze może prowadzić do narastania błędów w wyniku kolejnych operacji arytmetycznych. Dlatego ważne jest, aby być świadomym ograniczeń reprezentacji liczb zmiennoprzecinkowych i ostrożnym w obliczeniach, zwłaszcza w przypadkach, gdzie dokładność jest krytyczna.

2. 5. Korzystając z poniższego programu sprawdź dokładność wykonywanych obliczeń, dla których prawidłowy wynik wynosi 137 (spróbuj zmienić kolejność wykonywanych operacji), sprawdź wszystkie możliwe kombinacje.

```
#include <iostream>
#include <iomanip>

int main() {
    double x1 = 1.00E+21;
    double x2 = 17.0;
    double x3 = -10.0;
    double x4 = 130.0;
    double x5 = -1.00E+21;

    // Kombinacja 1: Oryginalna kolejność
    double s1 = x1 + x2 + x3 + x4 + x5;
    std::cout << "s1: " << s1 << std::endl;

    // Kombinacja 2: Inna kolejność
    double s2 = x2 + x4 + x1 + x5 + x3;
    std::cout << "s2: " << s2 << std::endl;

    // Kombinacja 3: Jeszcze inna kolejność
    double s3 = x3 + x2 + x5 + x1 + x4;
    std::cout << "s3: " << s3 << std::endl;

    return 0;
}
```

```
s1: 0
s2: -10
s3: 130
```

Mamy 32 możliwe kombinacje, jednak już przy pierwszych trzech widzimy znaczną różnicę w wyniku co potwierdza obecność błędu obliczeniowego, który jest obecny przy każdych kalkulacjach tylko w większości przypadków jest na tyle mały że program jest w stanie go pominąć.

Wnioski:

Różne systemy liczbowe niosą ze sobą różne wady i zalety. Największą zaletą systemu dziesiętkowego jest jego czytelność, lecz wadą jego słaba kompatybilność z komputerami, które operują w systemach binarnych. Sytuacja z systemami binarnymi jest wręcz przeciwna czytelność jest znikoma, przy liczbach zapisanych na 64 lub już nawet 32 bitach ludzkie oko ma problem z wyłapaniem większych różnic pomiędzy liczbami. Dla liczb zmiennoprzecinkowych cechą charakterystyczną jest błąd obliczeniowy zazwyczaj niewielki jednak wraz ze zwiększeniem skali coraz bardziej zauważalny.