Wydział budowy maszyn i informatyki

Architektura komputerów

(Laboratorium No1)

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:

2. 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;
}
```

```
00000000000000000000000000101010        32 bits 42
01000000010010001111010111000011        32 bits 3.14000010490417480469
0100000000000101101111110000100110010101101010101111011110010000        64 bits
2.7182800000000002956
10000000000000000000000000000000        32 bits -2147483648
01111111111111111111111111111111        32 bits 2147483647
0000000000010000000000000000000000000000000000000000000000000000        64 bits
0.0000000000000000000
0111111111101111111111111111111111111111111111111111111111111111        64 bits
```

```
17976931348623157081452742373170435679807056752584499659891747680315726078002853876058955863276687817154045895351438246423432132688946418276846754670353751698604991057655128207624549009038932894407586850845513394230458323690322294816580855933212334827479782620414472316873817718091929988125040402618412485836800000000000000000000


...Program finished with exit code 0
Press ENTER to exit console.
```

Różne typy danych mają różne reprezentacje binarne, a reprezentacje zmiennoprzecinkowe (np. float i double) mogą być skomplikowane, co wpływa na dokładność i precyzję obliczeń. Reprezentacje binarne są niezależne od wartości, a ich zrozumienie jest istotne przy pracy z danymi na poziomie niskiego poziomu.

2. 2.Zamienić liczby całkowite dziesiętne 91, -100 oraz 60 na ich odpowiedniki w kodzie uzupełnieniowym do dwóch (U2) z wykorzystaniem 32 bitów (n=32) i następnie porównać z otrzymanymi wartościami z programu implementowanego w ramach instrukcji 1.

Dla liczby 91:

91 w kodzie binarnym: `00000000000000000000000001011011` Kod uzupełnieniowy do dwóch (U2): `00000000000000000000000001011011` Tożsame z wynikiem w programie. Dla liczby -100:

100 w kodzie binarnym: `00000000000000000000000001100100` Odwrócenie bitów (negacja): `11111111111111111111111110011011` Dodanie 1: `11111111111111111111111110011100` Kod uzupełnieniowy do dwóch (U2): `11111111111111111111111110011100` Tożsame z wynikiem w programie. Dla liczby 60:

60 w kodzie binarnym: `00000000000000000000000000111100` Kod uzupełnieniowy do dwóch (U2): `00000000000000000000000000111100` Tożsame z wynikiem w programie. Podsumowując, przekształcone wartości do kodu uzupełnieniowego do dwóch (U2) są identyczne z wynikami otrzymanymi w programie dla odpowiednich wartości int. To pokazuje, że program poprawnie wyświetla reprezentacje binarne liczb całkowitych i można zaufać wynikom w programie.

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

```cpp
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 oczekiwana wartoscia." << std::endl;
    } else {
        std::cout << "Wynik nie jest zgodny z oczekiwana 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 oczekiwana wartoscia.

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

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