Laboratorium 1
Architektura Systemów Komputerowych
Temat 1 Systemy kodowania liczb ze znakiem i bez znaku

Zadanie 1

Korzystając z poniższego programu wyświetl zawartość pamięci dla różnych typów danych i porównaj binarne reprezentacje danych, a w szczególności sprawdź typy int, float oraz double definiując parametry o różnych wartościach. Dodatkowo zawartość pamięci dla maksymalnych i minimalnych wartości dla różnych typów. (Uwaga! Zmiany dokonujemy w głównej funkcji *main*)

```
#include <iostream>
#include <bitset>
#include <limits>
#include <iomanip>

template<typename T>
void show_32_bits(T tmp) {
  const unsigned int tmpSize =32;
  unsigned long int bits = *(unsigned long int*) (&tmp);
  std::bitset<(tmpSize)> tmpBits(bits);
  std::cout<<tmpBits <<"\t"<<tmpSize <<" bits"<<"\t"<<std::fixed<<std::setprecision(20)<<tmp<<std::endl;
}

template<typename T>
void show_64_bits(T tmp) {
  const unsigned int tmpSize = 64;
  unsigned long long int bits = *(unsigned long long int*) (&tmp);
  std::bitset<(tmpSize)> tmpBits(bits);
  std::cout<<tmpBits <<"\t"<<tmpSize <<" bits"<<"\t"<<std::fixed<<std::setprecision(20)<<tmp<<std::endl;
}

int main() {
  //modyfykujemy poniższy kod

  int a = -91;
  show_32_bits(a);

  int b = 91;
  show_32_bits(b);

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

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

  double e = -100;
  show_64_bits(e);

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

  return 0;
}
```

Kompilacja:

g++ nazwa_programu.cpp -o program_exe

./program_exe

Zadanie 2

Zamienić całkowite liczby dziesiętne 91, -100 oraz 60 na liczby binarne zapisane w kodzie uzupełnieniowym do dwóch (U2) z wykorzystaniem 32 bitów (n=32), a następnie porównaj z otrzymanymi wartościami z programu implementowanego w ramach instrukcji 1.

Zadanie 3

Sprawdź poprawność działania poniższego programu, a następnie spróbuj zmodyfikować program tak aby działał poprawnie.

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

int main() {

  double a = 0.1;
  double b = 0.2;
  double c = a + b;

  if(c == 0.3) {
    std::cout<<"suma="<<c<<std::endl;
  }

  return 0;
}
```

Zadanie 4

Korzystając z poniższego programu sprawdź dokładność wykonywanych obliczeń. W poniższym programie pętla wykona 100 iteracji, w ramach których sumaryczna wartość zmiennej *suma* zwiększa się za każdym razem o wartość 0.1 Jaki wynik powinien się wyświetlić, a jaki wyświetli się rzeczywiście?

```
#include <iostream>
#include <iomanip>
int main() {
  float suma = 0.0;
  for(int i=0; i<100; ++i)
    suma+=0.1f;
  std::cout <<std::fixed<<std::setprecision(20);
  std::cout <<"suma="<<suma<<std::endl;
  return 0;
}
```

Zadanie 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;

  double s1 =  x1 + x2 + x3 + x4 + x5;

  std::cout<<"s1 "<<s1<<std::endl;

  return 0;
}
```

Zadanie 6 - nieobowiązkowe

Mając na uwadze fakt, że wartość pierwszego bitu oznacza znak dla typów zmiennoprzecinkowych spróbuj napisać program umożliwiający zmianę znaku na przeciwny bazując na bitowych operacjach przeznaczonych dla typu stałoprzecinkowego.

```
#include <iostream>
#include <bitset>
#include <limits>
#include <iomanip>


template<typename T>
void show_32_bits(T tmp)
{
  const unsigned int tmpSize =32;
  unsigned long int bits = *(unsigned long int*) (&tmp);
  std::bitset<(tmpSize)> tmpBits(bits);
  std::cout<<tmpBits <<"\t"<<tmpSize <<" bits"<<"\t"<<std::fixed<<std::setprecision(20)<<tmp<<std::endl;
}

template<typename T>
void show_64_bits(T tmp)
{
  const unsigned int tmpSize = 64;
  unsigned long long int bits = *(unsigned long long int*) (&tmp);
  std::bitset<(tmpSize)> tmpBits(bits);
  std::cout<<tmpBits <<"\t"<<tmpSize <<" bits"<<"\t"<<std::fixed<<std::setprecision(20)<<tmp<<std::endl;
}

int main() {

  float tmp = 5.0;
  show_32_bits(tmp);

  // powyższy float traktowany jako int
  unsigned long  int * tmp_as_int = (unsigned long int*) (&tmp);
  show_32_bits(*tmp_as_int);

  // zastanów się jaka operacja na bitach może zamienić znak,
  // oraz jaka wartość operanda dla zmiennej bit_1 będzie tutaj konieczna

  unsigned long  int bit_1 = ?????;
  show_32_bits(bit_1);

  unsigned long int new_as_int = (*tmp_as_int ????  bit_1);
  show_32_bits(new_as_int);

  //nasz nowy float wydobyty z "inta"
  float new_tmp = *(float*)(&new_as_int);
  show_32_bits(new_tmp);

  return 0;
}
```