Zadanie 1

```
#include <stdio.h>

float x_with_float(int n) {
    float acc = 0.01f;
    for (int i = 0; i < n; i++)
        acc = acc + 3.0f * acc * (1.0f - acc);
    return acc;
}

double x_with_double(int n) {
    double acc = 0.01;
    for (int i = 0; i < n; i++)
        acc = acc + 3.0 * acc * (1 - acc);
    return acc;
}


int main() {
   const int n = 500000;
   printf("n = %d | float:%f double:%f\n", n, x_with_float(n), x_with_double(n)
   return 0;
}
```

Wyniki

```
n = 500000 | float:0.086346 double:1.285257
```

Wyniki się rozbiegają, ponieważ obliczenia są prowadzone na zmiennych z różnymi precyzjami. W przypadku mniej dokładnego typu `float` (na lokalnej maszynie 32 bity) ciąg wydaje się zbiegać w kierunku `0`, a w przypadku podwójnej precyzji `double` (na lokalnej maszynie 64 bity) wydaje się rozbiegać.

Zadanie 2

```c
#include <stdio.h>

float x_with_float(int n) {
    float acc = 0.01f;
    for (int i = 0; i < n; i++)
        acc = acc + 3.0f * acc * (1.0f - acc);
    return acc;
}

double x_with_double(int n) {
    double acc = 0.01;
    for (int i = 0; i < n; i++)
        acc = acc + 3.0 * acc * (1.0 - acc);
    return acc;
}

float x_alternate_with_float(int n) {
    float acc = 0.01f;
    for (int i = 0; i < n; i++)
        acc = 4.0f * acc - 3.0f * acc * acc;
    return acc;
}

double x_alternate_with_double(int n) {
    double acc = 0.01;
    for (int i = 0; i < n; i++)
        acc = 4.0 * acc - 3.0 * acc * acc;
    return acc;
}


int main() {
   const int n = 500000;
   printf("x{n} + 3.0 * x{n} * (1 - x{n}) | n = %d | float:%f double:%f\n", n,
   printf("4.0 * x{n} - 3.0 * x{n} * x{n} | n = %d | float:%f double:%f\n", n,
   return 0;
}
```

Wyniki

```
x{n} + 3.0 * x{n} * (1 - x{n}) | n = 500000 | float:0.086346 double:1.285257
4.0 * x{n} - 3.0 * x{n} * x{n} | n = 500000 | float:0.664632 double:1.059040
```

Wyniki drugiego obliczenia są do siebie bardziej zbliżone niż poprzednio, mimo że oba wzory są matematycznie jednoznaczne. Może to wynikać z faktu, że pierwsze obliczenie korzystało z wartości pośredniej `1 - x{n}`, a drugie korzysta wyłącznie bezpośrednio z `x{n}`.

Zadanie 3

Wartości epsilon (najmniejszej wartości takiej, że `1 + e > 1`) szukamy metodą biisekcji.sekcji.

```c
#include <stdio.h>

double find_epsilon_with_double() {
    double e = 0;
    double prev_e = 1;
    double mid = 0;

    const int iterations = 100000;
    for (int i = 0; i < iterations; i++) {
        mid = e + (prev_e - e) / 2.0;
        if(mid + 1 > 1)
            prev_e = mid;
        else e = mid;
    }

    return prev_e;
}

int main() {
   const double epsilon = find_epsilon_with_double();
   printf("Epsilon is: %e\n", epsilon);
   return 0;
}
```

Wyniki

```
Epsilon is: 1.110223e-16
```

Zwiększenie liczby iteracji nie powoduje dalszych zmian w wyniku, więc dotarliśmy do "maszynowego epsilona".