Для облегчения понимания, я вам написал небольшое пояснение, как хранятся десятичные дроби в памяти машины. Это поможет вам на контрольной, так как будут соответствующие задачи.

В современных ЭВМ вещественные числа представляются как числа с плавающей точкой (так достигается бОльшая точность по сравнению с числами с фиксированной точкой; легче хранить очень большие и очень маленькие числа). При этом двоичные дроби приводятся к нормализованной форме,
т.е. вида `±mE(±p)`, где `m` - мантисса, число `1 ≤ m < 10` (в двоичной СС), а `p` - порядок.
Эта запись означает `±m * 2^(±p)` (если рассматриваются двоичные числа).
В зависимости от того, какой тип данных нам нужен (**короткий**, **длинный** или **расширенный**), выбираем, сколько бит у нас отводится на мантиссу и сколько на порядок.
Тут очень удобно воспользоваться таблицей из 1-й презентации Коротина, слайда 41.

- Ну, например, нам надо представить в **коротком** виде число **6.7**.

  1. Для начала переводим его в двоичную дробь
     (отдельно целую часть $6_{10} = 110_2$, отдельно дробную по известному алгоритму:
     0.7 * 2 = 1.4 - выписываем в ответ целую часть, т.е. 1, а дробную продолжаем умножать на 2
     0.4 * 2 = 0.8 - дописываем в ответ 0
     0.8 * 2 = 1.6
     0.6 * 2 = 1.2
     0.2 * 2 = 0.4
     0.4 * 2 = 0.8
     0.8 * 2 = 1.6 - зациклились...
     выписывая целые части результатов, получаем, что $6.7_{10} = 110.10110(0110)_2$)

  2. Теперь приводим это число к ***нормализованной*** форме (т.е. сдвигаем точку к старшей единичке): $110.10110(0110)_2 = 1.1010110(0110)_2E10_2$ (здесь 10_2, стоящая после E - тоже в двоичном виде, т.е. это попросту 2_{10}).

  3. В **коротком** представлении (ему соответствует Сишный тип float, 4 байт) под порядок отводится **8** бит, 1 бит под знак числа, всё остальное (**23** бита) - под мантиссу.

  4. Далее. Порядок во всех трёх представлениях записывается в ***смещённом*** виде, т.е.
     число 0 записывается как 0111...11,
     число 1 как 1000...00,
     число 2 как 1000...01,
     число -1 как 0111...10,
     число -2 как 0111...01, и т.д.
     Т.е. отталкиваемся от нуля (0111...11) и вычисляем, как кодируется данный порядок.
     В нашем примере получается 1000...01.

  5. После битов порядка следом записываются биты мантиссы, причём лидирующие "1." не учитываются (старший бит в нормализованном виде всегда равен 1, поэтому на нём экономят).

  Резюмируя всё вышесказанное, записываем число $\mathbf{6.7}_{10}$ в двоичном виде с плавающей точкой в **коротком** представлении так:

```
знак / порядок / мантисса
    0100 0000 1101 0110 0110 ...          - в  2-чном виде.
      4    0    D    6    6    6  6  6    - в 16-чном виде.
```

  Именно это мы и увидим в отладчике-дизассемблере, если для кода `float f; scanf("%f", &f);` введём число 6.7.

- Далее. Для **длинного** представления (тип double, 8 байт) под порядок выделяется **11** бит, под мантиссу **52** бита.

```
знак /  порядок   / мантисса
    0100 0000 0001 1010 1100 1100 ...            - в  2-чном виде.
      4    0    1    A    C    C  C  C  CCCCCCCD  - в 16-чном виде.
```

  Последняя тетрада получила значение на 1 больше из-за округления.

- Наконец, для **расширенного** представления (тип long double, 10 байт) под порядок выделяется **15** бит, под мантиссу **64**.
  Только для этого типа старший бит мантиссы "1." **не** отбрасывается.

```
знак /    порядок      / мантисса
    0100 0000 0000 0001 1101 0110 0110 ...             - в  2-чном
виде.
      4    0    0    1    D    6    6  6  666666666666  - в 16-чном
виде.
```

  Далее ещё надо не забыть про обратный порядок байт (little-endian).
  Т.е. получившееся число в регистре процессора, например,
  401ACCCCCCCCCCCD,
  в оперативной памяти или в файле на диске будет храниться в перевёрнутом виде (с точностью до БАЙТ, но не БИТ!):
  CDCCCCCCCCCC1A40.
  Именно это мы и увидим, если в дизассемблерном отладчике, например, M$ Visual Studio, откроем кусок памяти, начиная с адреса `&d`, где `double d = 6.7`.

Посмотрите ещё две разобранные задачи с контрольной работы, там разобраны ещё пару примеров переводов туда и обратно.

Комментарии

Имя:

Комментарий:

Отправить