Задание на практическое занятие 3

1. Написать программу `utf8FromUnicode`, которая принимает один десятичный целочисленный беззнаковый параметр `code` — код символа unicode, и печатает на стандартый вывод беззнаковое десятичное целое число, которое в двоичном представлении соответствует двоичному представлению закодированного в utf8 значения `code`. При реализации придерживаться условия, что код символа в utf8 не будет превышать четырёх байт. Например:

   Запуск:

   ```
   utf8FromUnicode 1049
   ```

   Значение 1049 в десятичной системе счисления соответствует коду символа `й`.

   Вывод:

   ```
   53409
   ```

   Двоичное представление:

   ```
   1101000010011001
   ```

   Для реализации будет удобно следовать следующему алгоритму:

   1. Преобразовать параметр командной строки при запуске в целое число.
   2. Определить в какой диапазон входит значение параметра для определения того, в скольких байтах будет представлен символ в utf8. Это можно посмотреть в слайдах, которые лежат рядом с этим заданием.
   3. Определить сколько младших байтов требуется для представления, последовательно вырезать из разрядной сетки кода символа нужное количество шестибитовых значений для младших байтов и записать их в соответствующие байты результата (перед этим нужно не забыть прибавить в вырезанным значениям число `0x80` — это выставит старшие разряды младших байтов в значение `10` в двоичной системе счисления).
   4. В зависимости от количества байт в коде символа определить значение шаблона старшего байта. Если символ в utf8 заниает два байта, то шаблоном будет шестнадцатеричное число `0xc0`, если три байта — `0xe0`, если четыре — `0xf0`.
   5. Записать оставшиеся биты в старший байт, старший байт записать в результат.

   После каждого шага удобно будет выводить результат для контроля за ходом алгоритма. До начала реализации алгоритма подготовьте несколько значений кодов символов и вычислите соответствующие им коды в utf8.

Для тестирования и демонстрации следует использовать программу, печатающую двоичное представление целого числа, реализованную в предыдущей практической работе. Чтобы передать то, что `utf8FromUnicode` печатает на стандартный вывод в качестве параметра командной строки программе, выводящей двоичное представление своего параметра, следует использовать программу `xargs`. Например:

```
./utf8FromUnicode 1049 | xargs ./printBin
```

Результат запуска:

```
00000000000000001101000010011001
```

Здесь используется оператор конвейера `|`, который захватывает стандартный вывод программы `utf8FromUnicode` и передаёт его на стандартный ввод `xargs`, которая запускает программу `printBin` со значением, которое пришло в качестве параметра на стандартный ввод. Т.е., если `utf8FromUnicode` запускалась с параметром `1049`, то `xargs` запустит `printBin` с параметром `53409`.

2. Реализовать программу `unicodeFromUtf8`, которая принимает в качестве параметра целое беззнаковое число — закодированный символ в utf8. И возвращает целое беззнаковое число — код символа unicode.

   Для реализации удобно будет следовать следующим шагам:

   1. Определить количество байт, которые занимает код символа в utf8.
   2. В зависимости от количества байт, вырезать значимые биты из старшего и младших байтов.
   3. Последовательно склеить их в одно значение.

   ```
   ./unicodeFromUtf8 53409
   ```

   Результат запуска:

   ```
   1049
   ```

3. Написать программу `utf16FromUnicode`, которая принимает один десятичный целочисленный беззнаковый параметр `code` — код символа unicode, и печатает на стандартый вывод беззнаковое десятичное целое число, которое в двоичном представлении соответствует двоичному представлению закодированного в utf16 значения `code`.

   При реализации программы удобно будет следовать следующему алгоритму:

   1. Если codepoint находится в диапазоне от 0 до 0xFFFF (включительно), то он кодируется одной 16-битной кодовой единицей (просто число в двоичном виде, дополненное до 16

бит).
2. Если codepoint больше или равен 0x10000, то мы используем суррогатную пару:
    1. Вычтем 0x10000 из codepoint, получим 20-битное число (от 0 до 0xFFFFF).
    2. Разделим эти 20 бит на старшие 10 бит и младшие 10 бит.
    3. Добавим 0xD800 к старшим 10 битам, чтобы получить первую часть пары (старший суррогат).
    4. Добавим 0xDC00 к младшим 10 битам, чтобы получить вторую часть пары (младший суррогат).
3. Затем мы должны представить результат в виде 32-битного числа — в случае кодирования суррогатными парами в переменную типа `int` в старшие 16 бит записывается старший суррогат, а в младшие 16 бит записывается младший суррогат.

```
./utf16FromUnicode 66 615
```

Результат запуска:

```
3624942647
```

4. Реализовать программу `unicodeFromUtf16`, которая принимает в качестве параметра целое беззнаковое число — закодированный символ в utf16. И возвращает целое беззнаковое число — код символа unicode.

   Для реализации программы удобно будет использовать следующий алгоритм:

   1. Если число <= 0xFFFF (и не является суррогатом), то это BMP символ -> возвращаем число.
   2. Если число > 0xFFFF, то предполагаем, что это суррогатная пара (32 бита).

   Для суррогатной пары:

   1. Разделим число на старшие 16 бит (high) и младшие 16 бит (low).
   2. Проверим, что high в диапазоне 0xD800..0xDBFF и low в диапазоне 0xDC00..0xDFFF.
   3. Если да, то вычисляем code point

```
./unicodeFromUtf16 3624942647
```

Результат запуска:

```
66551
```

5. Реализовать программу `floatToBin`, которая принимает один параметр — вещественное число, и печатает на стандартный вывод его двоичное представление.

   Для преобразование параметра из строки в число можно воспользоваться функцией `atof`:

```
float num = atof(argv[1]);
```

При реализации программы, удобно будет следовать следующему алгоритму:

1. Перевести параметр командной строки в вещественное число.
2. Сохранить в отдельную переменную целую часть числа. Чтобы взять только целую часть следует воспользоваться приведением к типу `int` — `int m = (int)num`. В `m` будет находиться целая часть вещественного числа `num`.
3. Сохранить в отдельную вещественную переменную дробную часть числа. Чтобы это сделать нужно из исходного вычесть целую часть.
4. Перевести двробную часть в двоичное представление, которое хранить в переменной типа `int`. Для этого можно воспользоваться алгоритмом перевода умножением и записью целой части числа в ответ:
    1. Объявить переменную типа `int` в которой будет храниться результат. И переменную в которой будет подсчитано количество бит результата.
    2. Умножить дробь на 2.
    3. Сдвинуть результат побитово влево на одну позицию.
    4. Прибавить к результату результат умножения дроби на два, приведённого к типу `int` (для того чтобы сохранить в результат только целую часть). Увеличить переменную в которой хранится количество бит на 1.
    5. Избавиться от целой части результата умножения дробной части на 2. Чтобы это сделать нужно вычесть из результата умножения на 2 результат умножения приведённый к типу `int`.
    6. Если после вычитания дробь не равна нулю и суммарное количество значимых разрядов целой и дробной части не больше 24, перейти на шаг 2, иначе закончить перевод.
5. Определить порядок нормализованного числа. Для этого нужно определить сколько значимых разрядов в целой части:
    1. Если больше одного — порядок будет равен `количество разрядов - 1`
    2. Если равно одному, то порядок равен нулю.
    3. Иначе, нужно определить на какой позиции двоичного представления дробного числа находится старший единичный разряд и из позиции вычесть количество разрядов дробного представления
6. Закодировать порядок
7. Записать в переменную типа `int` двоичное представление вещественного числа, путём выставления на нужные позиции двоичного представления битов знака, закодированного порядка и мантиссы.