Курс лекций "Программирование"

Основы программирования на языках C и C++

Лекция 4. Методы хранения данных в ЭВМ

Глухих Михаил Игоревич, к.т.н., доц.

mailto: glukhikh@mail.ru

Системы счисления

- Система счисления – символический метод записи чисел; говоря иначе, способ представления чисел с помощью цифр (знаков, символов)
- Какие системы счисления бывают?

Десятичная система счисления

- Примеры записи
  - $35164_{10}(=3*10^4+5*10^3+1*10^2+6*10+4)$
  - $6.48_{10}=6+4*10^{-1}+8*10^{-2}$
- Позиционная система счисления – вес каждой цифры зависит от ее положения (налево от десятичной точки – 1, 10, 10^2 и т.д.; направо – 10^{-1}, 10^{-2}, 10^{-3} и т.д.)

Позиционные p-ичные системы счисления

- p - натуральное число не равное 1 – основание системы счисления
- Числа представляются в форме:
  - $X=x_n*p^n+x_{n-1}*p^{n-1}+\ldots+x_1*p+x_0+ x_{-1}*p^{-1}+\ldots$, где x_i – цифры числа X в p-ичной системе; справедливо $0<=x_i<p$
- Сокращенная форма записи
  - $X=x_nx_{n-1}x_1x_0.x_{-1}x_{-2}\ldots$
- Например, $25_{10}=121_4$, так как $2*10+5=1*4^2+2*4+1$

Какие системы счисления используются чаще других?

- **Люди** в основном используют десятичную систему счисления. Основная причина – число пальцев на руках (удобно считать десятками)
- **Компьютеры**, как правило, хранят числа в **двоичной** системе счисления. Операции над числами также делаются в двоичной системе счисления. При выводе числа переводятся из двоичной системы в десятичную, при вводе – из десятичной в двоичную

Почему именно двоичная система?

- Элементы с двумя четко выраженными состояниями проще в использовании и существенно дешевле, чем элементы с тремя и более состояниями
- Основные операции в двоичной системе выполняются легче (ввиду простоты таблиц сложения и умножения)

Перевод целых чисел из 10-ичной системы в p-ичную

- Начиная с младших разрядов
- Пример для p=5, X=183_{10}
  - 183 | 5 → **3**
  - 36 | 5 → **1**
  - 7 | 5 → **2**
  - 1 | 5 → **1**
  - 0
- Итог: $183_{10}=1213_5$

Перевод целых чисел из 10-ичной системы в p-ичную

- Начиная со старших разрядов
- Пример для p=5, X=183_{10}
  - 183 >= 125 (5^3)
  - 183/125=**1**, 183-1*125=58
  - 58/25=**2**, 58-2*25=8
  - 8/5=**1**, 8-1*5=3
  - 3/1=**3**, 3-1*3=0
- Итог: $183_{10}=1213_5$

Перевод дробных чисел из 10-ичной системы в p-ичную

- Пример для p=7, x=0.23
  - 0.23 * 7 → **1** + 0.61
  - 0.61 * 7 → **4** + 0.27
  - 0.27 * 7 → **1** + 0.89
  - 0.89 * 7 → **6** + 0.23
  - …
- Итог: $0.23_{10} \sim 0.1416_{7}$

Перевод чисел из p-ичной системы в десятичную

- Обратный перевод: $1213_5=???_{10}$
- $1213_5=1*5^3+2*5^2+1*5+3=$
  $=125+50+5+3=183$
- Обратный перевод: $0.1416_7=???_{10}$
- $0.1416_7=1/7+4/49+1/343+6/2401=$
  $=0.1429+0.0816+0.0029+0.0025=$
  $=0.2299\sim0.23$

Из k-ичной системы в n-ичную (k и n не равны 10)

- Как правило, переводят через десятичную систему счисления. Впрочем, есть исключения – ситуации, когда $k=n^m$ или наоборот

Как прочитать данные в двоичной системе счисления

- Может понадобиться при чтении машинных кодов команд, когда важен каждый конкретный двоичный разряд (**бит**)
- Однако двоичное число может занимать очень много места, и легко ошибиться при поиске в нем нужной цифры
- Поэтому на практике в таких случаях используют шестнадцатеричную систему или (реже) восьмеричную систему

Шестнадцатеричная система счисления

- Шестнадцать цифр: 0-9, A-F (или a-f)
- Каждая 16-ичная цифра соответствует в точности четырем двоичным, так как $16=2^4$. Например, $3_{16}=0011_2$, $b_{16}=1011_2$
- Перевод чисел 2→16 или 16→2 может производиться непосредственно. При этом четыре 2-ичных цифры меняются на одну 16-ичную (начиная от точки), и наоборот. Например:
  - $1101101101_2=36d_{16}$; $0.0101110_2=0.5c_{16}$

Шестнадцатеричная система счисления (язык C/C++)

- Константы: впереди добавляется 0x, например:
  - **int** hexNumber = 0xa96d;
- Вывод: используется манипулятор hex, например:
  - cout<<hex<<hexNumber<<endl;
- Будет выведено a96d (без 0x)
- Для возврата к десятичной системе используется манипулятор dec
- Оба механизма работают только с целыми числами

Восьмеричная система счисления

- Восемь цифр: 0-7
- Каждая 8-ичная цифра соответствует в точности трем двоичным, так как $8=2^3$. Например, $4_8=100_2$, $6_8=110_2$
- Перевод чисел 2→8 или 8→2 может производиться непосредственно. При этом три 2-ичных цифры меняются на одну 8-ичную (начиная от точки), и наоборот. Например:
  - $1101101101_2=1555_8$; $0.0101110_2=0.270_8$

Восьмеричная система счисления (язык C/C++)

- Константы: впереди добавляется 0, например:
  - **int** octNumber = 0733;
- Вывод: используется манипулятор oct, например:
  - cout<<oct<<octNumber<<endl;
- Будет выведено 733 (без 0)
- Для возврата к десятичной системе используется манипулятор dec
- Оба механизма работают только с целыми числами

Пример. Перевод дробных чисел в двоичную систему.

```
#include <iostream>
#include <locale.h>
using namespace std;
int main(void)
{
   setlocale(LC_ALL, "Russian");
   cout<<"Преобразование в двоичную систему"<<endl;
   double x;
   do
   {
      cout<<"Введите вещ. число 0<x<1: ";
      cin>>x;
   } while (x<=0 || x>=1);
   cout<<"Двоичное число: 0.";
```

Пример. Перевод дробных чисел в двоичную систему.

```
    for (int i=0; i<24; i++)
    {
        x *= 2;
        // Выводим и вычитаем целую часть
        cout<<(int)x;
        x -= (int)x;
        // Если число слишком маленькое,
        // остальные цифры =0
        if (x <= 1e-8)
            break;
    }
    cout<<endl;
    return 0;
}
```

Как хранятся числа в памяти ЭВМ?

- Ячейки памяти состоят из отдельных двоичных разрядов.
- **Бит** – один двоичный разряд – единица измерения количества информации
- Восемь (2^3) **бит** образуют **байт**
- Сколько байт в килобайте?

Как хранятся числа в памяти ЭВМ?

- Ячейки памяти состоят из отдельных двоичных разрядов.
- **Бит** – один двоичный разряд – единица измерения количества информации
- Восемь (2^3) **бит** образуют **байт**
- Сколько байт в килобайте?
- 2^{10}=1024 байт образуют **килобайт** (Кбайт). Почему не 1000? Чтобы машине было удобнее считать ячейки
- Аналогично, 2^{10} Кбайт=1 **Мегабайт**, 2^{10} Мбайт=1 **Гигабайт**, 2^{10} Гбайт=1 **Терабайт**

Сколько бит занимает целое число (int)?

- Процессор ЭВМ при выполнении операций использует целые числа, содержащие определенное число двоичных разрядов (бит)
- Большинство современных процессоров оперируют 32-разрядными целыми числами (поэтому их называют 32-разрядными процессорами)
- Поэтому в современных средах разработки стандартный целочисленный тип (int) также занимает 32 двоичных разряда

Каков диапазон беззнаковых 32-разрядных целых чисел?

- Максимально возможное число равно $1+2+2^2+2^3+\ldots+2^{30}+2^{31}=2^{32}-1\sim4*10^9$
- Тип беззнаковых целых чисел в C/C++ обозначается **unsigned int** или просто **unsigned**, например:
  - **unsigned** bigNumber=0xf4e10026;
- Беззнаковый тип используется сравнительно редко (обычный **int** считается более эффективным по скорости работы).

А как хранятся целые числа со знаком?

- Старший разряд выделяется под знак числа. Причем, 0 соответствует “+”, 1 соответствует “-”
- Обычно используется **дополнительный** код:
  - Для положительных чисел в оставшихся разрядах (31) хранится |x|
  - Для отрицательных чисел в оставшихся разрядах хранится 2^{32}-|x|. Например, -17 хранится как 1111 1111 1111 1111 1111 1111 1110 1111 (можно проинвертировать все разряды модуля и добавить 1 в младший разряд)
- Зачем так сложно??? Почему не хранить |x| для отрицательных чисел тоже?
  - Такой способ есть. Он называется **прямой** код.
  - Лучше, однако, дополнительный. Потому что числа в дополнительном коде можно складывать так же, как беззнаковые

Пример для 8-разрядных чисел

$+93 \rightarrow +1011101_2 \rightarrow 01011101_{доп}$

$-119 \rightarrow -1110111_2 \rightarrow 10001001_{доп}$

```
      11  1
   01011101
   10001001
   --------
```

$11100110_{доп} \rightarrow -0011010_2 \rightarrow -26$

Поэтому.

- Диапазон стандартных целых чисел в C/C++ (тип int) от -2^{31} до $2^{31}-1$
- Что будет, если к максимальному целому прибавить единицу?
  - **int** maxInt=0x7fffffff;
  - cout<<maxInt+1<<endl;
- На экран выведется -2147483648 (что соответствует -2^{31})

Целые типы с другим диапазоном

- **short** или **short int** – обычно 16-разрядный тип (-32768…32767), используется обычно, когда необходимо сохранить большое количество однотипных данных для экономии памяти
- **long** или **long int** – в большинстве современных реализаций C/C++ совпадает с **int** – 32 разряда; в 64-разрядных системах 64 разряда
- **long long** – 64-разрядный тип ($-2^{63}…2^{63}-1$), используется, когда не хватает 32 разрядов
- У всех указанных типов есть беззнаковая **unsigned** версия

Побитовые логические операции

- Выполняются **отдельно** над каждым разрядом (битом) целого числа (**bool**, **short**, **int**, **long**)
- Обычно используются при хранении в целочисленном формате списка логических свойств (например, шрифта)
  - & побитовое логическое И (равно 1, если оба аргумента равны 1)
  - | побитовое логическое включающее ИЛИ (равно 1, если хотя бы один аргумент равен 1)
  - ^ побитовое логическое исключающее ИЛИ (равно 1, если один аргумент равен 1, а другой 0)
- Если мы работаем с целыми числами, равными 0 или 1 (или с типом **bool**), то побитовые операции аналогичны обычным

Пример использования побитовых операций

```
…
const int FONT_BOLD = 0x1;
const int FONT_ITALIC = 0x2;
const int FONT_UNDERLINE = 0x4;
int someFont = FONT_BOLD | FONT_ITALIC;
int anotherFont;
// Какие-то операции с anotherFont
…
if (anotherFont & FONT_UNDERLINE)
{
    …
}
…
```

Операции поразрядного сдвига

- Сдвигают целое число в двоичной форме вправо (в сторону младших разрядов) или влево (в сторону старших разрядов). Количество разрядов указывается вторым аргументом (целое положительное число не более общей разрядности).
  - сдвиг вправо >>
  - сдвиг влево <<
- При сдвиге влево младшие разряды заполняются нулями. Сдвиг влево на 1 разряд аналогичен умножению на 2 (при этом, он может выполняться быстрее).
- При сдвиге вправо старшие разряды заполняются знаком. Сдвиг вправо на 1 разряд аналогичен делению на 2 (и он также может выполняться быстрее).

Примеры работы

…

```
int someNumber = 0x36; // 110110
// 110110 << 2 = 11011000 (0xc8)
int anotherNumber = someNumber << 2;
int negativeNumber = -9076;
// -9076 / 16 = -568
negativeNumber >>= 4;
```

Пример. Перевод целых чисел в двоичную систему.

```
#include <iostream>
#include <locale.h>
using namespace std;
int main(void)
{
   setlocale(LC_ALL, "Russian");
   cout<<"Преобразование в двоичную систему"<<endl;
   int num;
   do
   {
      cout<<"Введите целое число >0: ";
      cin>>num;
   } while (num<=0);
   cout<<"Двоичное число: ";
   int binValue=0x40000000; // значение дв-го разряда
```

Пример. Перевод целых чисел в двоичную систему.

```
    // Делим на 2, пока не получим меньшее число
    while (binValue > num) binValue >>= 1;
    while (binValue>0)
    {
        if (num >= binValue)
        {
            cout<<"1";
            num -= binValue;
        }
        else
            cout<<"0";
        binValue >>= 1;
    }
    cout<<endl; return 0;
}
```

Как хранятся вещественные числа?

- Используется форма **с плавающей точкой** $m*2^p$, где m – **мантисса**, p – **порядок**
- Мантисса нормируется: 0.5<=m<1, в ней хранятся двоичные цифры после точки и знак
- Порядок – обычное целое число со знаком
- Пример: $0.03125_{10}=0.5*2^{-4}$ → m=0.1, p=-100

Сколько разрядов занимают мантисса и порядок?

- Используется стандарт IEEE 754
- Числа с одинарной точностью (**float**) – 8 разрядов порядок, 24 разряда мантисса, всего 32 разряда, точность 7 десятичных разрядов, максимум около $3.4*10^{38}$
- Числа с двойной точностью (**double**) – 11 разрядов порядок, 53 разряда мантисса, всего 64 разряда, точность 15 десятичных разрядов, максимум около $1.7*10^{308}$
- Иногда встречается еще тип **long double** (80 разрядов), но в MVSC++ он совпадает с **double**

Организация памяти в ЭВМ

- Используется иерархический принцип: маленькая и быстрая память для часто используемых данных, большая и медленная для редко используемых
  - регистры процессора – несколько 32-разрядных очень быстрых ячеек
  - кэш, используется для ускорения доступа к оперативной памяти, обычно нескольких уровней, например, 64Кб – 3Мб – 24Мб; в кэше нельзя ничего разместить непосредственно
  - оперативная память (единицы Гб)
  - жесткий диск (сотни Гб, единицы Тб)

Организация памяти при выполнении программы

- Стек (stack) – наращивается по мере появления новых переменных, уменьшается по мере их исчезновения. Содержит **автоматические** переменные. Размер по умолчанию 1 Мб (MSVC 2005).
- Статическая память (static allocation memory) – выделяется перед выполнением программы, очищается после ее окончания. Содержит машинные коды программы и **статические** переменные.
- Динамическая память (куча, heap, dynamic allocation memory) – выделяется и разрушается по требованию во время выполнения программы. **Динамические…** не переменные

Автоматические переменные в языках C/C++

- Все переменные, рассмотренные нами ранее, были автоматические и размещались в стеке
- Определяются внутри функций; можно (необязательно) перед названием типа добавить ключевое слово **auto**
- Размещаем в стеке, когда достигаем определения переменной
- Убираем из стека, когда достигаем конца блока
- Поэтому, могут использоваться между определением и концом блока, где определение находится

Пример работы со стеком

```
int main(void)
{
    int i,j; // Разместили i,j
    …
    // Разместили k
    for (int k=0; k<10; k++)
    {
        auto int n=72; // Разместили n
        …
        // Убрали n
    }
    // Убрали k
    return 0;
    // Убрали i,j
}
```

Регистровые переменные

- Перед названием типа переменной можно указать ключевое слово **register**
- При этом, переменная будет размещена в регистре, если будет свободный, или в стеке, если свободного регистра не будет
- Используется для увеличения производительности
- **for** (**register int** i=0; i<10; i++) …

Статические переменные

- Могут быть определены двумя способами
  - Вне функций

```
// статическая внешняя переменная
int stVar = 91;
int main(void)
{
}
```

  - Внутри функций, тогда **обязательно** добавление ключевого слова **static** перед названием типа

```
int main(void)
{
    // статическая внутренняя переменная
    static int stVar = 91;
    …
}
```

Статические переменные

- Вечные – существуют, пока выполняется программа
- Однако, могут быть недоступны в каких-то ее участках

Внешние файлы

- Используются для нескольких целей:
  - Хранение настроек программы
  - Хранение входных данных программы
  - Хранение документов, баз данных и другой информации, редактируемой или читаемой программой
  - Хранение промежуточных данных, используемых при работе
  - Хранение выходных данных

Расширение файла

- Расширение файла указывает на его формат, например:
  - .txt – текстовый формат
  - .bin или .dat – двоичный формат
  - .ini – формат настроечных файлов
  - .xml – формат XML (extensible markup language)
  - и так далее

Использование файлов в языках C и C++

```
#include <iostream>
#include <fstream>
#include <locale.h>
using namespace std;
int main(void)
{
   setlocale(LC_ALL, "Russian");
   // Открытие входного файла in.txt
   ifstream in("in.txt"); // Определяем переменную in
   if ( !in.is_open() ) // Успешно ли открытие
   {
      cout<<"Ошибка: файл in.txt не существует"<<endl;
      return -1;
   }
```

Использование файлов в языках C и C++

```
int a, b, c;
// Ввод трех целых чисел из файла
in>>a>>b>>c;
if ( in.fail() ) // Не произошла ли ошибка
{
   cout<<"Ошибка: числа не прочитаны"<<endl;
   return -2;
}
cout<<"Прочитаны числа: "<<a<<", "<<
      b<<", "<<c<<endl;
```

Использование файлов в языках C и C++

```
    // Открытие выходного файла out.txt
    ofstream out("out.txt");
    // Определяем переменную out
    if ( !out.is_open() )
    {
        cout<<"Ошибка: не удается записать "<<
              "файл out.txt"<<endl;
        return -3;
    }
    // Вывод a+b+c=sum в выходной файл
    out<<a<<"+"<<b<<"+"<<c<<"="<<a+b+c<<endl;
    cout<<"Программа успешно завершена!"<<endl;
    return 0;
}
```

Основные функции и типы при работе с файлами

- <fstream> – подключаемый файл
- ifstream – тип «входной файл»
- ofstream – тип «выходной файл»
- Далее in – имя переменной типа ifstream
- in.is_open() – был ли файл открыт
- in.fail() – была ли ошибка при последней операции
- in.eof() – закончился ли файл

Итоги

- Рассмотрели
  - системы счисления
  - способы хранения чисел
  - организация памяти
  - файлы
- Далее
  - массивы
  - строки
  - указатели