Лабораторная работа 1 (0001 = 1)

Представление данных в ЭВМ

Цель работы: изучить размеры стандартных типов C/C++ и форматы представления чисел на выбранной платформе.

Все задания данной лабораторной работы выполняются на чистом C/C++, без использования ассемблера.

Штраф за одно пропущенное обязательное задание:
-1 балл, если лабораторная работа сдаётся на первом занятии;
-2 балла — если на втором и последующих.

Задание на лабораторную работу

Задание Л1.з1. Выберите платформу, на которой планируется в дальнейшем выполнять лабораторные работы.

Укажите:
- ОС и разрядность ОС;
- компилятор (должен относиться к коллекции GCC/MinGW) и его версию;
- разрядность сборки (собираемая программа может работать в 32-битном режиме даже под 64-битной ОС — в режиме совместимости);
- архитектуру процессора, назначение платформы.

Примечание: компьютер с процессором x86/x86-64 под управлением GNU/Linux, BSD (в том числе Mac OS X) или MS Windows — платформа общего назначения.

Задание Л1.з2. При помощи оператора `sizeof` языка C/C++ выясните, сколько байтов занимают переменные следующих типов языка C/C++: `char`, `signed char`, `unsigned char`, `wchar_t`, `short`, `unsigned short`, `int`, `unsigned int`, `long`, `unsigned long`, `long long`, `unsigned long long`, `float`, `double`, `long double`, `size_t`, `ptrdiff_t`, `void*`, `char*`, `int*`, `unsigned int*` на выбранной платформе.

Обратите внимание на размеры целочисленных типов и чисел с плавающей запятой. Какие из них имеют разрядность 16, 32, 64 бита, учитывая, что байт x86/x86-64 — октет (8 бит)?

Задание Л1.з(3−2ε). Бонус +2 балла для НБ-31, +0 для прочих.

На языке C/C++ разработайте функцию void $viewPointer(\text{void } *p)$, которая принимает нетипизированный указатель p, преобразует его в типизированные:

а) `char *p1 = reinterpret_cast<char *>(p);`
б) `unsigned short *p2 = reinterpret_cast<unsigned short *>(p);`
в) `double *p3 = reinterpret_cast<double *>(p);`

и печатает $p, p1, p2, p3$ (не значения по этим адресам, а сами адреса).

Проверьте работу функции $viewPointer()$ на адресах нескольких переменных типов $char, double, int$.

Убедитесь, что $p, p1, p2, p3$ — один и тот же адрес, то есть что оператор `reinterpret_cast` не меняет преобразуемого указателя и, следовательно, может быть использован для интерпретации одной и той же области памяти как значений различных типов.

Примечание: Обратите внимание на то, что при печати указателя $p1$ (на однобайтовый целый тип) результат отличается от $p, p2, p3$ — так как в C/C++ не определён строковый тип, вместо строк используются массивы однобайтовых целых и основанные на них классы.

Соответственно, оператор << для указателей на типы *char* / *unsigned char* / *int8_t* / *uint8_t* перегружен и печатает не адрес, а значения — начиная с указанного адреса и до ближайшего нулевого байта.

Дополните $viewPointer()$ печатью смежных с p адресов: $p + 1$ (если текущие настройки компилятора позволяют рассчитать такое выражение), $p1 + 1$, $p2 + 1$, $p3 + 1$. Сопоставьте разницу между p_i и $p_i + 1$ в байтах для типизированного указателя $T * p_i$ с размером типа T.

Задание Л1.з(3−ε). Бонус +2 балла для НБ-31, +0 для прочих.

На языке C/C++ разработайте функцию void $printPointer$(void $* p$), которая принимает нетипизированный указатель p, преобразует его в типизированные $p1, p2, p3$ аналогично $viewPointer()$ и печатает значения различных типов по адресу p: $*p1, *p2, *p3$. Можно ли рассчитать (и, соответственно, напечатать) $*p$?

Все целые числа выводите в шестнадцатеричном виде (для чего в начале $main()$ напечатайте манипулятор hex). Проверьте работу функции $printPointer()$ на значениях `0x8877665544332211` (*long long*), `"abcdefgh0123456789"` (*char*[]).

Дополните $printPointer()$ печатью значений по смежным с p адресам: $*(p1 + 1), *(p2 + 1), *(p3 + 1)$.

Задание Л1.з3. Изучите, как интерпретируется одна и та же область памяти, если она рассматривается как знаковое или беззнаковое целое число, а также — как одно и то же число записывается в различных системах счисления.

Для этого на языке C/C++ разработайте функцию void $print16$(void $* p$), которая печатает для 16-битной области памяти по заданному адресу p:

а) целочисленную беззнаковую интерпретацию в шестнадцатеричном представлении;
б) целочисленную беззнаковую интерпретацию в двоичном представлении;
в) целочисленную беззнаковую интерпретацию в десятичном представлении;
г) целочисленную знаковую интерпретацию в шестнадцатеричном представлении;
д) целочисленную знаковую интерпретацию в двоичном представлении;
е) целочисленную знаковую интерпретацию в десятичном представлении.

Примечание: для получения различных интерпретаций одного и того же участка памяти в C++ можно использовать объединения (`union`) или преобразование *указателя p в указатель на другой тип* оператором `reinterpret_cast` или приведением в стиле C. Обратите внимание, что преобразование *значения в значение* оператором `static_cast` или приведением в стиле C не обеспечивает необходимого эффекта (хотя для целых типов одно-

го размера $signed \leftrightarrow unsigned$ преобразование значения в значение чаще всего приводит к тому же результату, что разыменованный преобразованный указатель).

Имена беззнаковых целочисленных типов C++ содержат ключевое слово $unsigned$ (так, беззнаковый тип того же размера, что и $short$, называется $unsigned\ short$). Имена знаковых целочисленных типов могут содержать ключевое слово $signed$ либо никакого (так, $signed\ short$ и $short$ — синонимы).

Соответственно, целочисленную беззнаковую интерпретацию памяти по адресу p для (а) и (в) можно получить, как `*(reinterpret_cast<unsigned short *>(p))`, знаковую для (г) и (е) — как `*(reinterpret_cast<short *>(p))` (см. разделы 7.2 и 7.3).

Шестнадцатеричное и десятичное представление целых чисел можно получить, используя различные форматы вывода функции $printf()$ библиотеки libc либо манипуляторы hex и dec потокового вывода.

Шестнадцатеричный формат вывода для целочисленных переменных соответствует компактной записи двоичного кода, поэтому совпадает для беззнаковой и знаковой интерпретаций (и равен беззнаковой интерпретации в шестнадцатеричной системе счисления).

На вывод чисел с плавающей запятой манипуляторы hex, oct, dec никакого влияния не оказывают.

Двоичное (битовое) представление чисел можно получить, используя шаблон `std::bitset<N>`, где N — количество бит в представлении — необходимо задать вручную.

Проверьте работу функции $print16()$ на 16-битных целочисленных переменных, принимающих следующие значения:

- минимальное целое 16-битное значение без знака;
- максимальное целое 16-битное значение без знака;
- минимальное целое 16-битное значение со знаком;
- максимальное целое 16-битное значение со знаком;
- значение y, соответствующее варианту (таблица Л1.1);
- значение z, соответствующее варианту (таблица Л1.1);

(запишите каждое из значений в 16-битную целочисленную переменную и передайте её адрес функции).

Убедитесь, что (а) и (г) — одно и то же шестнадцатеричное представление; аналогично, (б) и (д) — одно и то же двоичное представление.

Полученные результаты внесите в отчёт в таблицу, каждая строка которой соответствует значению, столбец — представлениям (а), (б), (в), (е).

Задание Л1.з4. Разработайте на языке C/C++ функции $print32()$ и $print64()$, аналогичные $print16()$ для размеров 32 и 64 бита, и дополните их интерпретацией памяти как числа с плавающей запятой («вещественного» числа) соответствующего размера (для 32 бит — с одинарной точностью, $float$; для 64 — с двойной точностью, $double$). Необходимо напечатать:

ж) интерпретацию с плавающей запятой в представлении с фиксированным количеством цифр после запятой;

з) интерпретацию с плавающей запятой в экспоненциальном представлении.

Варианты значений

Таблица Л1.1

$(№ - 1)\%2 + 1$	**Вариант**
1	$x =$ `0xF1F2F3F4`, $y = 4$, $z = -7$
2	$x =$ `0x8A8B8C8D`, $y = 6$, $z = -3$

Примечание: для вывода в поток обратите внимание на манипулятор $setprecision()$ и метод $setf()$.

Проверьте работу функций на целочисленных переменных соответствующего размера, принимающих значения:

- минимальное целое значение без знака соответствующего размера;
- максимальное целое значение без знака соответствующего размера;
- минимальное целое значение со знаком соответствующего размера;
- максимальное целое значение со знаком соответствующего размера;
- значение x, соответствующее варианту;
- значение y, соответствующее варианту;
- значение z, соответствующее варианту;

и переменных с плавающей запятой соответствующего размера, принимающих значения:

- значение x, соответствующее варианту;
- значение y, соответствующее варианту;
- значение z, соответствующее варианту;

значения x, y, z смотрите в таблице Л1.1. Выпишите в отчёт полученные результаты (дополните таблицу задания Л1.з3 столбцами (ж) и (з).

Задание Л1.з5. Изучите, как располагаются в памяти байты, составляющие целое число и число с плавающей запятой. Для этого на языке C/C++ разработайте функцию void $printDump$(void $*\, p$, size_t N), которая печатает для области памяти по заданному адресу p значения N байтов, начиная с младшего, в шестнадцатеричном представлении (шестнадцатеричный дамп памяти).

С помощью $printDump()$ определите и выпишите в отчёт, как хранятся в памяти компьютера в программе на C/C++:

- целое число x (типа int; таблица Л1.1); по результату исследования определите порядок следования байтов в словах для вашего процессора:
  а) прямой (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);

   б) обратный (младший байт по старшему адресу, порядок Motorola, Big-Endian, от старшего к младшему);
- массив из трёх целых чисел (статический или динамический, но не высокоуровневый контейнер) с элементами x, y, z;
- число с плавающей запятой y (типа $double$; таблица Л1.1).

Примечание: однобайтовые целочисленные переменные (не только $char$ / $unsigned\ char$, но и $int8\_t$ / $uint8\_t$) выводятся в поток *в виде символа,* код которого равен значению переменной (независимо от манипуляторов $hex/oct/dec$). Для вывода в поток значения однобайтовой переменной в виде шестнадцатеричного, десятичного или восьмеричного числа необходимо расширить это значение до двух- или более байтового типа ($char/int8\_t$ в $short$, int и т. п.; $unsigned\ char/uint8\_t$ — в $unsigned\ short$, $unsigned$ и т. п.).

Для преобразования *значения в значение* в C++ используется оператор `static_cast` (или универсальное, но не рекомендованное приведение в стиле C).

Л1.1. Дополнительные бонусные и штрафные баллы

+1 балл за корректную (то есть такую, что с макросами или шаблонами код короче или хотя бы красивее, чем был бы без них) автоматизацию с помощью макросов препроцессора C или шаблонов C++.

−3 балла за утечку памяти (то есть наличие выделенных, но не освобождённых блоков динамической памяти).

Л1.2. Ссылки на теоретические сведения

1.2.1. Единицы измерения
1.2.2. Порядок следования байтов
2.4. Двоичное представление беззнаковых целых чисел
2.5. Представление отрицательных чисел
2.8. Представление вещественных чисел
7.2. Типы данных
7.3. Приведение типов
7.4. Литералы C++
7.5. Средства автоматизации C++
4.2. Препроцессор
7.6. Ввод-вывод
7.7. Отладочная печать

Л1.3. Вопросы

1. Как представляются целые числа со знаком и без знака?
2. Как перевести число в дополнительный код?
3. Для чего нужно знать порядок следования байтов на вашем компьютере?
4. Как располагаются в памяти элементы массива?

5. Как найти размер массива, зная размер элемента и их количество?