ПИН-22, команда 5: Надёжкина Анна, Трусов Михаил

Отчет

Лабораторная работа 2 (0010 = 2)

Представление данных в ЭВМ

Операционная система ОС Windows, 64-разрядная ОС.

Задание Л2.з1.

*Разработайте функцию void viewPointer(void * p), которая принимает нетипизированный указатель p, преобразует его в типизированные: а) char *p1 = reinterpret_cast(p); б) unsigned short *p2 = reinterpret_cast(p); в) double *p3 = reinterpret_cast(p); и печатает p, p1, p2, p3 (не значения по этим адресам, а сами адреса). 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. Дополните viewPointer() печатью смежных с p адресов: p1 + 1, p2 + 1, p3 + 1. Сопоставьте разницу между pi и pi + 1 в байтах для типизированного указателя T * pi с размером типа T. Проверьте, позволяют ли текущие настройки компилятора рассчитать p + 1. Если да — какова разница между p и p + 1 в байтах*

Программный код:

```
#include <stdio.h>

using namespace std;


void viewPointer(void *p) {
  // размер char - один байт
  char *p1 = reinterpret_cast<char *>(p);
  // размер unsigned short - два байта
  unsigned short *p2 = reinterpret_cast<unsigned short *>(p);
  // размер double - восемь байт
  double *p3 = reinterpret_cast<double *>(p);


  printf("Address p: %p\n", p);
  printf("Address p1: %p\n", p1);
  printf("Address p2: %p\n", p2);
  printf("Address p3: %p\n", p3);


  printf("\nAddress p1+1: %p\n", p1 + 1);
  printf("Address p2+1: %p\n", p2 + 1);
```

```
  printf("Address p3+1: %p\n", p3 + 1);
}

int main() {
  int x = 5;
  void *p = &x;
  viewPointer(p);

  return 0;
}
```

Вывод:

```
Address p: 0x7ffffcbe4
Address p1: 0x7ffffcbe4
Address p2: 0x7ffffcbe4
Address p3: 0x7ffffcbe4

Address p1+1: 0x7ffffcbe5
Address p2+1: 0x7ffffcbe6
Address p3+1: 0x7ffffcbec
```

Задание Л2.з2.

*Разработайте функцию void printPointer(void *p), которая принимает нетипизированный указатель p, преобразует его в типизированные p1, p2, p3 аналогично viewPointer() и печатает значения соответствующих типов по адресу p: *p1, *p2, *p3. Можно ли рассчитать (и, соответственно, напечатать) *p? Дополните printPointer() печатью значений по смежным с p адресам: *(p1 + 1), *(p2 + 1), *(p3 + 1). Все целые числа выводите в шестнадцатеричном виде. Проверьте работу функции printP ointer() на значениях 0x1122334455667788 (long long), "0123456789abcdef" (char[])*

Нельзя прямо разыменовывать указатель типа void*, так как void не имеет размера и не позволяет получить доступ к конкретному значению, так как компилятору неизвестен конкретный тип данных, на который указывает void*. То есть, попытка разыменовывать указатель типа void* вызовет ошибку компиляции. Если необходимо получить доступ к значению, хранящемуся по адресу, указанному в void*, его необходимо явно привести к соответствующему типу перед дальнейшим использованием.

Программный код:

```
#include <stdio.h>
using namespace std;

void printPointer(void *p) {
  char *p1 = reinterpret_cast<char *>(p);
  unsigned short *p2 = reinterpret_cast<unsigned short *>(p);
  double *p3 = reinterpret_cast<double *>(p);

  printf("\nZnachenie p1: %c\n", *p1);
  printf("Znachenie p2: %04hX\n", *p2);
  printf("Znachenie p3: %.2le\n", *p3);

  printf("\nZnachenie p1+1: %c\n", *(p1 + 1));
  printf("Znachenie p2+1: %04hX\n", *(p2 + 1));
  printf("Znachenie p3+1: %.2le\n", *(p3 + 1));
}

int main() {
  long long x = 0x1122334455667788;
  printPointer(&x);

  char y[] = "0123456789abcdef";
  printPointer(&y);
  return 0;
}
```

Вывод:

Znachenie p1: ▒

Znachenie p2: 7788

Znachenie p3: 3.84e-226

Znachenie p1+1: w

Znachenie p2+1: 5566

Znachenie p3+1: 1.70e-313

Znachenie p1: 0

Znachenie p2: 3130

Znachenie p3: 9.96e-43

Znachenie p1+1: 1

Znachenie p2+1: 3332

Znachenie p3+1: 1.82e+185

Задание Л2.з3. Вариант 1 ($x = 9, y = -9, z = 0x88776155$).

*Разработайте функцию void printDump(void * p, size_t N), которая принимает нетипизированный указатель p, преобразует его в типизированный указатель на байт unsigned char * p1 и печатает шестнадцатеричные значения N байтов, начиная с этого адреса: *p1, *(p1 + 1), ... * (p1 + (N – 1)) — шестнадцатеричный дамп памяти. Каждый байт должен выводиться в виде двух шестнадцатеричных цифр; байты разделяются пробелом.*

С помощью printDump() определите и выпишите в отчёт, как хранятся в памяти компьютера в программе на C/C++: – целое число x (типа int; таблица Л2.1); 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, BigEndian, 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 Л2.1).

Прямой порядок следования байтов (младший байт по младшему адресу, порядок Intel, Little-Endian, от младшего к старшему);

Программный код:

```
#include <stdio.h>

using namespace std;


void printDump(void* p, size_t N) {
  unsigned char* p1 = reinterpret_cast<unsigned char*>(p);
  for (size_t i = 0; i < N; ++i) {
    printf("%02hhx ", *(p1 + i));
```

```
  }
  printf("\n");
}

int main() {
  int x = 9;
  printDump(&x, sizeof(x));

  int arr[3];
  arr[0] = 9;
  arr[1] = -9;
  arr[2] = 0x88776155;
  printDump(&arr, sizeof(arr));

  double y = -9;
  printDump(&y, sizeof(y));
  return 0;
}
```

Вывод:

09 00 00 00

09 00 00 00 f7 ff ff ff 55 61 77 88

00 00 00 00 00 00 22 c0

Задание Л2.з4. Вариант 1 ($x = 9$, $y = -9$, $z = 0\text{x}88776155$).

*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 области памяти по заданному адресу p: а) целочисленную беззнаковую 16-битную интерпретацию (unsigned short) в шестнадцатеричном представлении; б) целочисленную беззнаковую 16-битную интерпретацию (unsigned short) в двоичном представлении; в) целочисленную беззнаковую 16-битную интерпретацию (unsigned short) в десятичном представлении (для printf() используйте формат u: она умеет выводить любые целые в любом представлении, и требуется*

явное указание); г) целочисленную знаковую 16-битную интерпретацию (short) в шестнадцатеричном представлении; д) целочисленную знаковую 16-битную интерпретацию (short) в двоичном представлении; е) целочисленную знаковую 16-битную интерпретацию (short) в десятичном представлении (для printf() используйте формат d).

Бонус +1 балл, *если вывод print16() занимает одну строку (так на экран поместится больше чисел).* **Бонус +2 балла**, *если при этом младшая цифра находится под младшей цифрой предыдущей строки (чего можно добиться заданием ширины поля вывода).*

Программный код:

```
#include <stdio.h>


#include <bitset>

#include <iomanip>

#include <iostream>


using namespace std;


void print16(void *p) {

    unsigned short *a = reinterpret_cast<unsigned short *>(p);

    printf("%04X ", *a);

    printf("%16s ", bitset<16>(*a).to_string().c_str());

    printf("%5u ", *a);


    short *b = reinterpret_cast<short *>(p);

    printf("%04X ", *b&0xFFFF);

    printf("%16s ", bitset<16>(*b).to_string().c_str());

    printf("%6d \n", *b);

}



int main() {

    // Минимальное целое 16-битное значение без знака

    unsigned short min_unsigned_short = 0;

    print16(&min_unsigned_short);
```

```
    // Максимальное целое 16-битное значение без знака
    unsigned short max_unsigned_short = 65535;
    print16(&max_unsigned_short);

    // Минимальное целое 16-битное значение со знаком
    short min_short = -32768;
    print16(&min_short);

    // Максимальное целое 16-битное значение со знаком
    short max_short = 32767;
    print16(&max_short);

    short x = 9;
    print16(&x);

    short y = -9;
    print16(&y);

    return 0;
}
```

Вывод:

```
0000 0000000000000000     0 0000 0000000000000000      0
FFFF 1111111111111111 65535 FFFF 1111111111111111     -1
8000 1000000000000000 32768 8000 1000000000000000 -32768
7FFF 0111111111111111 32767 7FFF 0111111111111111  32767
0009 0000000000001001     9 0009 0000000000001001      9
FFF7 1111111111110111 65527 FFF7 1111111111110111     -9
```

Задание Л2.з5. Вариант 1 ($x = 9$, $y = -9$, $z = 0x88776155$).

*Разработайте на языке C/C++ функцию void $print32$(void *p), аналогичную $print16$() для размера 32 (каждое из дублирующихся представлений — шестнадцатеричное (а) и (г), двоичное (б) и (д) — выводить один раз). Кроме целочисленных интерпретаций, $print32$() должна рассматривать память по адресу p как 32-битное число с плавающей запятой («вещественное») одинарной*

точности (float) и печатать: ж) 32-битную интерпретацию с плавающей запятой (float) в представлении с фиксированным количеством цифр после запятой; з) 32-битную интерпретацию с плавающей запятой (float) в экспоненциальном представлении.

Программный код:

```
#include <stdio.h>


#include <bitset>

#include <iomanip>

#include <iostream>


using namespace std;


void print32(void *p) {
  unsigned int *a = reinterpret_cast<unsigned int *>(p);
  printf("%08X ", *a);
  printf("%32s ", bitset<32>(*a).to_string().c_str());
  printf("%10u ", *a);


  int *b = reinterpret_cast<int *>(p);
  printf("%11d ", *b);


  float *c = reinterpret_cast<float *>(p);
  printf("%13.2f ", *c);
  printf("%5.2e \n", *c);
}


int main() {
  unsigned int min_unsigned_int = 0;
  print32(&min_unsigned_int);


  unsigned int max_unsigned_int = 4294967295;
  print32(&max_unsigned_int);
```

```
  int min_int = -2147483648;
  print32(&min_int);

  int max_int = 2147483647;
  print32(&max_int);

  int x = 9;
  print32(&x);

  int y = -9;
  print32(&y);

  int z = 0x88776155;
  print32(&z);

  float xx = 9;
  print32(&xx);

  float yy = -9;
  print32(&yy);

  float zz = 0x88776155;
  print32(&zz);

  return 0;
}
```

Вывод:

```
00000000 00000000000000000000000000000000           0           0          0.00 0.00e+00
FFFFFFFF 11111111111111111111111111111111  4294967295          -1          -nan  -nan
80000000 10000000000000000000000000000000  2147483648 -2147483648         -0.00 -0.00e+00
7FFFFFFF 01111111111111111111111111111111  2147483647  2147483647           nan   nan
00000009 00000000000000000000000000001001           9           9          0.00 1.26e-44
FFFFFFF7 11111111111111111111111111110111  4294967287          -9          -nan  -nan
88776155 10001000011101110110000101010101  2289525077 -2005442219         -0.00 -7.44e-34
41100000 01000001000100000000000000000000  1091567616  1091567616          9.00 9.00e+00
C1100000 11000001000100000000000000000000  3239051264 -1055916032         -9.00 -9.00e+00
4F087761 01001111000010000111011101100001  1325954913  1325954913 2289524992.00 2.29e+09
```

Задание Л2.з6. Бонус +2 балла. Вариант 1 ($x = 9, y = -9, z = 0x88776155$).

Разработайте на языке C/C++ функцию print64(), аналогичную print32() для размера 64 бита и, соответственно, числа с плавающей запятой двойной точности, double. Аналогично Л2.з5, проверьте print64() на граничных целочисленных 64-битных значениях, целочисленных значениях x, y, z и double-значениях x, y, z.

Бонус +1 балл, *если вывод print32() (и print64(), если она реализована) занимает одну строку.*
Бонус +2 балла, *если при этом младшая цифра находится под младшей цифрой предыдущей строки.*

Программный код:

```
#include <stdio.h>


#include <bitset>

#include <iomanip>

#include <iostream>


using namespace std;


void print64(void *p) {

  unsigned long long *a = reinterpret_cast<unsigned long long *>(p);

  printf("%016X ", *a);

  printf("%64s ", bitset<64>(*a).to_string().c_str());

  printf("%10u ", *a);


  long long *b = reinterpret_cast<long long *>(p);

  printf("%11d ", *b);


  double *c = reinterpret_cast<double *>(p);

  printf("%13.2f ", *c);
```

```
  printf("%5.2e \n", *c);
}

int main() {
  // Минимальное целое 64-битное значение без знака
  unsigned long long min_unsigned_longlong = 0;
  print64(&min_unsigned_longlong);

  // Максимальное целое 64-битное значение без знака
  unsigned long long max_unsigned_longlong = 18446744073709551615;
  print64(&max_unsigned_longlong);

  // Минимальное целое 64-битное значение со знаком
  int min_longlong = -9223372036854775808;
  print64(&min_longlong);

  // Максимальное целое 64-битное значение со знаком
  int max_longlong = 9223372036854775807;
  print64(&max_longlong);

  int x = 9;
  print64(&x);

  int y = -9;
  print64(&y);

  int z = 0x88776155;
  print64(&z);

  float xx = 9;
  print64(&xx);
```

```
  float yy = -9;
  print64(&yy);

  float zz = 0x88776155;
  print64(&zz);

  return 0;
}
```

Вывод: