Структурное программирование

ЛЕКЦИЯ №2

12 СЕНТЯБРЯ 2023

Альтернативные варианты хранения двоичных чисел

- Пользователи привыкли к десятичной системе (не 2, 8 или 16).
- Преобразование между 2 и 10-й системами требует деления и умножения. (Для небольших встроенных систем это может быть критично)

Варианты?

Альтернативные варианты хранения двоичных чисел

- Пользователи привыкли к десятичной системе (не 2, 8 или 16).
- Преобразование между 2 и 10-й системами требует деления и умножения. (Для небольших встроенных систем это может быть критично)

Варианты?

- Хранить число строкой, каждая цифра отдельный символ.
- Двоично-десятичное кодирование (Binary-Coded Decimal, BCD), каждая цифра кодируется 4 битами независимо.

BCD

Пример кодирования числа 337:

{3}{3}{7} ->

BCD

Пример кодирования числа 337:

{3}{3}{7} -> {0011}{0011}{0111} =

Т.е. десятичное число 337 кодируется в BCD как в двоичном представлении.

Для знака обычно выделяется младший полубайт: «+» кодируется как = «-» кодируется как

Итого 337 кодируется как 1100 0011 0111 1100

В 32 битах представляются числа -9 999 999…9 999 999

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

, коррекция младшего разряда:

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

, коррекция младшего разряда:

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

, коррекция младшего разряда:

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

, коррекция младшего разряда:

, коррекция старшего разряда:

Операции с BCD

Можно делать сложение и вычитание битовых представление BCD чисел, если выполнять коррекцию при переполнениях: добавлять или вычитать 6 из соответствующих разрядов.

, нет переполнения, все ок.

, коррекция младшего разряда:

, коррекция старшего разряда:

Преимущества BCD

Преимущества BCD

- Упрощенный ввод/вывод
- Нет потерь точности при вводе/выводе вещественных чисел (об этом позже).
- Легко умножать/делить на 10, округлять до десятичных разрядов.

Битовые операции

Битовые операции

Стандартный набор битовых логических операций включает в себя:

- Логическое И &
- Логическое ИЛИ |
- Исключающее ИЛИ ^
- Отрицание ~
- Битовые сдвиги (вправо >>) и (влево <<)

Логическое И

X	Y	X & Y (X AND Y)
0	0	0
0	1	0
1	0	0
1	1	1

Логическое умножение

Например для чисел:
00100011 (35) и 00001111 (15)

Логическое ИЛИ

X	Y	X | Y (X OR Y)
0	0	0
0	1	1
1	0	1
1	1	1

Логическое сложение без переноса в следующий разряд

Например для чисел:
00100011 (35) и 00001111 (15)

Исключающее ИЛИ

X	Y	X ^ Y (X XOR Y)
0	0	0
0	1	1
1	0	1
1	1	0

Логическое сложение без переноса в следующий разряд

Например для чисел:
00100011 (35) и 00001111 (15)

Отрицание

X	~X (NOT X)
0	1
1	0

Логическое сложение без переноса в следующий разряд

Например для числа:
00100011 (35)

Битовые сдвиги

Лишние биты – отбрасываем.

Новые биты зануляем. (при сдвиге вправо – не всегда ☹)

Сдвиг влево << синтаксис a<<n, где n – на сколько позиций сдвигать.

Пример 00100011<<2 (35):

10001100 (140) (если число без знака)

Как быстро посчитать результат сдвига?

Битовые сдвиги

Лишние биты – отбрасываем.

Новые биты зануляем.

Сдвиг влево << синтаксис a<<n, где n – на сколько позиций сдвигать.

Пример 00100011<<2 (35):

10001100 (140) (если число без знака)

Сдвиг на 1 бит = умножение на 2

Битовые сдвиги

Лишние биты – отбрасываем.

Новые биты зануляем.

Сдвиг вправо << синтаксис a<<n, где n – на сколько позиций сдвигать.

Пример 00100011>>2 (35):

00001000 (15)

Как быстро посчитать?

Битовые сдвиги

Лишние биты – отбрасываем.

Новые биты зануляем.

Сдвиг вправо << синтаксис a<<n, где n – на сколько позиций сдвигать.

Пример 00100011>>2 (35):

00001000 (15)

Сдвиг на 1 бит =деление на 2 (остаток отбрасывается).

Практическое применение битовых операций

Чтение бита:

Практическое применение битовых операций

Чтение бита:

1&(x>>i) 00000001&(10110111>>2) Итог: 00000001 (1)

Практическое применение битовых операций

Чтение бита:

1&(x>>i) 00000001&(10110111>>2) Итог: 00000001 (1)

Выставление бита:

Практическое применение битовых операций

Чтение бита:

1&(x>>i) 00000001&(10110111>>2) Итог: 00000001 (1)

Выставление бита:

x|(1<<i) 10110111 |(00000001<<3) Итог: 10111111

Практическое применение битовых операций

Чтение бита:

1&(x>>i) 00000001&(10110111>>2) Итог: 00000001 (1)

Выставление бита:

x|(1<<i) 10110111 |(00000001<<3) Итог: 10111111

Сброс бита:

Практическое применение битовых операций

Чтение бита:

1&(x>>i) 00000001&(10110111>>2) Итог: 00000001 (1)

Выставление бита:

x|(1<<i) 10110111 |(00000001<<3) Итог: 10111111

Сброс бита:

x&(~(1<<i)) 10111111 |~(00000001<<3) Итог: 10110111

Хранение символов

Хранение символов в памяти

Символы для отображения пользователю необходимо каким-то образом записать в виде двоичного кода.

Кодировка – сопоставление между символом и его кодом.

ASCII — фундаментальная 7-битная кодировка:

- 33 управляющих символа,
- 95 информационных символов.

```
0x20: ␣!"#$%&'()*+,-./
0x30: 0123456789:;<=>?
0x40: @ABCDEFGHIJKLMNO
0x50: PQRSTUVWXYZ[\]^_
0x60: `abcdefghijklmno
0x70: pqrstuvwxyz{|}~
```

Строка `"Hello!"` кодируется байтами:

`48 65 6C 6C 6F 21`.

Хранение символов в памяти

- Null, `0x00`, `\0` — пустой символ,
- Bell, `0x07`, `\a` — звуковой сигнал,
- Backspace, `0x08`, `\b` — возврат на шаг,
- Character Tabulation, `0x09`, `\t` — горизонтальная табуляция,
- Line Feed, `0x0A`, `\n` — перевод строки,
- Line Tabulation, `0x0B`, `\v` — вертикальная табуляция,
- Form Feed, `0x0C`, `\f` — смена страницы,
- Carriage Return, `0x0D`, `\r` — возврат каретки.
- ...

Тип данных в си

char

char a='q';

Для символов одинарные кавычки!

'a'

Больше символов?

А что с кириллицей, умляутами и иероглифами?

Вариант: 8-битные кириллические кодировки, базирующиеся на ASCII: КОИ-8, CP866, Windows-1251, ...

На практике как-то так: РџСЂРёРІРµС‚, РјРёСЂ!

Юникод — многобайтовая кодировка, покрывающая почти все письменные языки.

Позволяет кодировать 1 112 064 символов. В версии 13.0, март 2020 г., используется лишь 143 859.

Содержит все национальные символы, математические символы, символы древних письменностей и многое другое, включая 😀 😐 ☹ 😂 😹 😄 😇 😊 😈 😉 😎 🐶 и даже 😸!

Составные типы данных: Массивы

Массивы

Набор однотипных данных фиксированной длины:

- Вектор – массив значений.
- Матрица – массив векторов/двумерный массив значений.
- Группа – массив студентов.

Одномерные массивы

Элементы массива в памяти находятся строго друг за другом.

Адрес i-го элемента:

Где - адрес нулевого элемента, S – размер одного элемента.

Адресом массива называют адрес его первого элемента.

Двумерные массивы

Массив 2x3 (в общем случае (M x N):

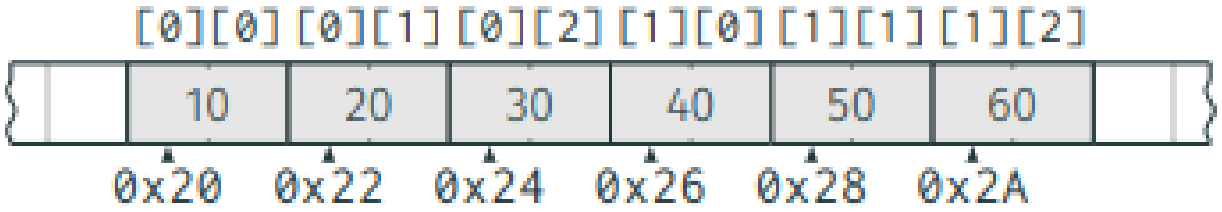
	0	1	2
0	0	1	2
1	3	4	5

Элементы все еще лежат в памяти подряд.

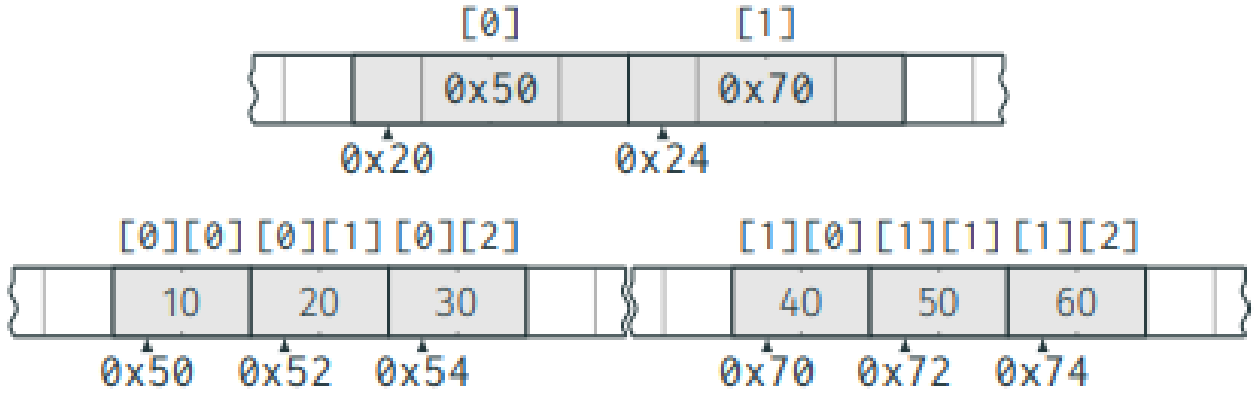
Элемент с номером [i][j] находится в позиции [3i+j]

Массив массивов

{{10, 20, 30}, {40, 50, 60}} в памяти выглядит как:



Однако, существует и иной подход:



Практическая реализация

Массивы в Си

Объявление и инициализация:

int A[5];

Int A[5]={1,2,3,4,5};

Int A[]={1,2,3,4,5};

Размер массива – константа компиляции!

Все операции с элементами очень простые:

Двумерные аналогично:

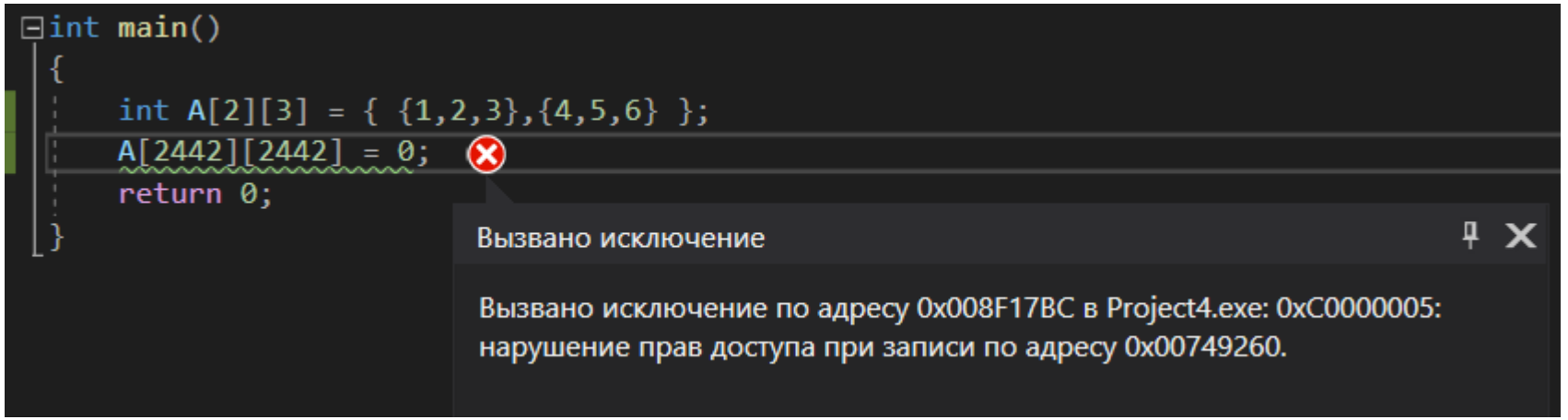
int A[2][3], int A[2][3]={{1,2,3},{4,5,6}}

Нельзя (НО НЕ ЗАПРЕЩЕНО) обращаться к элементам вне длины массива – программа не обязательно рухнет, но может произойти что-то непредвиденное.

Перед тем, как пытаться вывести значение элемента массива надо бы его записать (:

Плохие примеры

Попытка записи туда, куда не следует.



Еще один пример плохого поведения

Читаем неинициализированную память и пишем данные не в ту ячейку.

```c
int main()
{
    int A[22];
    int B[5];
    B[10] = 4;
    for (int i = 0; i < 22; i++)
    {
        printf("A[%d]=%d\n", i, A[i]);
    }
    return 0;
}
```

```
A[0]=-858993460
A[1]=-858993460
A[2]=-858993460
A[3]=4
A[4]=-858993460
A[5]=-858993460
A[6]=-858993460
A[7]=-858993460
A[8]=-858993460
A[9]=-858993460
A[10]=-858993460
A[11]=-858993460
A[12]=-858993460
A[13]=-858993460
A[14]=-858993460
A[15]=-858993460
A[16]=-858993460
A[17]=-858993460
A[18]=-858993460
A[19]=-858993460
A[20]=-858993460
A[21]=-858993460
```

Читаем и печатаем массив

```c
int main()
{
    int A[4];
    for (int i=0; i<4; i++)
    {
        printf("A[%d]=",i);
        scanf("%d", &A[i]);
    }

    for (int i = 0; i < 4; i++)
    {
        printf("A[%d]=%d\n", i, A[i]);
    }

    return 0;
}
```

```
A[0]=34
A[1]=6
A[2]=3
A[3]=2
A[0]=34
A[1]=6
A[2]=3
A[3]=2
```

Читаем и печатаем массив

```c
int main()
{
    int A[4];
    for (int i=0; i<4; i++)
    {
        printf("A[%d]=",i);
        scanf("%d", &A[i]);
    }

    for (int i = 0; i < 4; i++)
    {
        printf("A[%d]=%d\n", i, A[i]);
    }

    return 0;
}
```

```
A[0]=34
A[1]=6
A[2]=3
A[3]=2
A[0]=34
A[1]=6
A[2]=3
A[3]=2
```

```
A[0]=54
A[1]=4
A[2]=a
A[3]=A[0]=54
A[1]=4
A[2]=-858993460
A[3]=-858993460
```

Проверка ввода scanf

Функция scanf возвращает число успешно присвоенных значений.

Это можно и нужно использовать.

```c
void clearbuf()
{
    while (getchar() != '\n');
    return;
}
```

```c
int main()
{
    int A[4];
    for (int i=0; i<4; i++)
    {
        printf("A[%d]=",i);
        while (scanf("%d", &A[i]) != 1)
        {
            clearbuf();
            printf("Wrong input\n");
            printf("A[%d]=", i);
        }
    }

    for (int i = 0; i < 4; i++)
    {
        printf("A[%d]=%d\n", i, A[i]);
    }

    return 0;
}
```

Консоль отлад

```
A[0]=asg
Wrong input
A[0]=asg
Wrong input
A[0]=sheh
Wrong input
A[0]=4
A[1]=3
A[2]=he
Wrong input
A[2]=4
A[3]=5
A[0]=4
A[1]=3
A[2]=4
A[3]=5
```

На сегодня хватит