Язык программирования Си

Бикулов Д.А., Иваницкая Н.В., Иванов А.П.

Семинар 8. Пользовательские типы данных (enum, struct, union). Побитовые операторы.

1 Структуры

Структура — это объединение нескольких переменных под одним именем. При объявлении структуры можно задать тип структуры, который далее будет использоваться при создании экземпляров структуры. Объявляется структура с помощью ключевого слова **`struct`**. Доступ к полям (т.е. членам) структуры осуществляется с помощью оператора «**.**» (точка).

Пример. Создание структуры с типом **`MyStruct`**, содержащей поля **`a`**, **`word`**, **`name`** и **`number`**:

```
#include <stdio.h>

struct MyStruct {
    int a;
    char *word;
    char name[16];
    double number;
};

int main()
{
    MyStruct mst;          // MyStruct – имя типа
    Struct MyStruct mst2; // можно и так

    mst.a = 10;
    mst.word = "Hello!";
    strcpy(mst.name,"Alexey");
    mst.number = 4.8;

    printf("%d %s %lf\n", mst.a, mst.word, mst.number);

    return 0;
}
```

Можно сразу же после объявления структуры создать несколько экземпляров этого типа:

```
struct Student {
    int  age, group;
    char name[32], surname[32];
} student_pete, student_anna;
```

Размер структуры равен сумме размеров полей. Размер пустой структуры равен одному байту по стандарту, иначе возникли бы проблемы с адресацией памяти.

2 Перечисления

Объявление перечислений очень похоже на объявление структур. Отличие состоит в использовании ключевого слова **`enum`**. Перечисления представляют собой набор именованных целочисленных констант. Эти константы задают все возможные значения переменной объявленного типа.

Пример.

```
#include <stdio.h>

enum WeekDays {
    LunaeDies,
    MartisDies,
    MercuriiDies,
    JovisDies,
    VenerisDies,
    SaturniDies,
    SolisDies
};

int main()
{
    WeekDays wd = VenerisDies;

    printf("%d\n", wd);

    return 0;
}
```

В результате работы программы будет выведена цифра `4`. Переменная **`wd`** имеет тип **`WeekDays`** и может принимать только значения, явно указанные в перечислении. Если не указано ничего дополнительно, то константы в перечислении нумеруются, начиная с нуля.

Если для какой-либо константы указано явное значение, то нумерация продолжается дальше с этого значения, указывать значение можно для константы с любой позицией:

```
#include <stdio.h>

enum WeekDays {
    LunaeDies = 10,
    MartisDies,
    MercuriiDies,
    JovisDies,
    VenerisDies,
    SaturniDies = 20,
    SolisDies
};

int main()
{
    WeekDays wd[] = { LunaeDies, MartisDies, MercuriiDies, JovisDies,
                      VenerisDies, SaturniDies, SolisDies };

    for( int i=0; i <= sizeof(wd)/sizeof(wd[0]); i++ )
        printf("%d ", wd[i]);
    printf("\n");

    return 0;
}
```

В результате работы программы будут выведены числа:

```
10 11 12 13 14 20 21
```

3 Объединения

Объединения хранят значения всех своих полей в одной области памяти. В случае, если поля имеет разную длину, то выделяется память под наибольшее поле. Объединения нужны для экономия памяти в случае, если в разные неперекрывающиеся моменты времени требуется хранить массивы разных типов данных, имеющих одинаковую длину.

Пример. Объединения с полями различных типов:

```
#include <stdio.h>

union IntoFloat {
    float f;
    int i;
};

int main()
{
    IntoFloat uif;

    uif.i = 1000;
    printf("%d %f\n", uif.i, uif.f);
    uif.f = 10.5;
    printf("%d %f\n", uif.i, uif.f);

    return 0;
}
```

В результате работы программы будет выведено:

```
1000 0.000000
```

(второе значение – абсолютно случайный мусор)

```
1093140480 10.50000
```

(первое значение – абсолютно случайный мусор)

4 Побитовые операторы

Побитовые операции языка Си представлены пятью бинарными и одной унарной операцией. К побитовым операциям относятся:

- бинарная операция побитовое И: « **&** » (знак амперсанд);
- бинарная операция побитовое ИЛИ: « | » (знак вертикальная черта);
- бинарная операция побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ: « ^ » (знак карет);
- унарная операция побитовое отрицание, НЕ: « ~ » (знак тильды);
- бинарная операция побитовый СДВИГ ВПРАВО: « >> » (два знака больше);
- бинарная операция побитовый СДВИГ ВЛЕВО: « << » (два знака меньше).

Обратите внимание, формально операторы сдвига совпадают с операторами ввода-вывода языка Си++, которые использовались с самого начала этого курса. Но так как они применяются к другому типу данных левого операнда (к целому числу, а не к потоку ввода-вывода), то смысл их действия принципиально другой: они осуществляют побитовый сдвиг целого значения, указанного в первом операнде, вправо или влево на указанное во втором операнде целое число битов.

Таблица истинности для бинарных побитовых операций:

первый операнд	второй операнд	операция		
		& («И»)	**|** («ИЛИ»)	**^** «ИСКЛЮЧАЮЩЕЕ ИЛИ»
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Пример. Побитовые операции:

```
unsigned char a = 14;          // a=00001110  (14)
unsigned char b = 42;          // b=00101010  (42)

unsigned char c = ~a;          // c=11110001 (241)
c = a & b;                     // c=00001010  (10)
c = a | b;                     // c=00101110  (46)
c = a ^ b;                     // c=00100100  (36)
```

Пример. Операции сдвига:

```
unsigned char a = 11, b;       // a=00001011  (11)
b = a << 2;                    // b=00101100  (44)
b = a << 10;                   // b=00000000   (0)
b = a >> 1;                    // b=00000101   (5)
```

Все перечисленные операторы имеют дополнительный оператор «выполнить-присвоить», например:

```
unsigned char a = 10, b = 8;   // a=00001010  (10), b=00001000  (8)
a  &= b;                       // a=00001000   (8), b=00001000  (8)
a >>= 2;                       // a=00000010   (2)
```

Типовое задание: реализовать программу, обрабатывающую структуры данных, данные для работы считать из файла, результат – записать в файл. Например: считать из файла массив записей о студентах (ФИО, год рождения, номер группы), отсортировать записи по году рождения и записать результат в другой файл.

1. Вариант

Создать тип данных **рациональные числа** (числитель, знаменатель). Прочитать из файла массив рациональных чисел, отсортировать его с помощью библиотечной функции **`qsort()`**. Для использования функции **`qsort()`** создать свою функцию сравнения двух рациональных чисел.
Результат напечатать на экране специально созданной функцией.

2. Вариант

Создать тип данных **трехмерные вектора**. Прочитать из файла массив трехмерных векторов, отсортировать его с помощью библиотечной функции **`qsort()`** по убыванию нормы вектора. Для использования функции **`qsort()`** создать свою функцию сравнения норм двух векторов.
Результат напечатать на экране специально созданной функцией.

3. Вариант

Создать тип данных **музыкальные альбомы** (исполнитель, название, год выпуска). Прочитать из файла массив альбомов, отсортировать его с помощью библиотечной функции **`qsort()`** по исполнителям, а для одного исполнителя – по годам. Для использования функции **`qsort()`** создать свою функцию сравнения двух альбомов.
Результат вывести на экран в компактном виде: исполнитель, год, список альбомов в этом году.

4. Вариант

Создать тип данных **время** (чч:мм:сс). Прочитать из файла массив времен, отсортировать его с помощью библиотечной функции **`qsort()`** по возрастанию. Для использования функции **`qsort()`** создать свою функцию сравнения двух времен.
Результат напечатать на экране специально созданной функцией.

5. Вариант

Создать тип данных **календарная дата** (дд.мм.гггг), считать, что в месяце всегда ровно 30 дней. Прочитать из файла массив дат, отсортировать его с помощью библиотечной функции **`qsort()`** по убыванию разницы значения элемента массива с заданной пользователем датой. Для использования функции **`qsort()`** создать свою функцию сравнения.
Результат напечатать на экране специально созданной функцией.

6. Вариант

Создать тип данных **полиномы 4 порядка**. Записать коэффициенты полиномов, вводимых пользователем в бинарный файл. В отдельной программе запросить у пользователя число x, прочитать последовательно записанный бинарный файл и вывести на экран полином, значение которого на заданном x будет минимальным, а также значение этого полинома для заданного x.

7. Вариант

Создать тип данных **книги** (автор, заглавие, год выхода). Записать набор таких данных в бинарный файл записей фиксированной длины в отсортированном по автору и году виде. Написать функцию двоичного поиска всех произведений заданного пользователем автора в этом бинарном файле.
Результат напечатать на экране.

8. Вариант

Создать тип данных **комплексные полиномы 3 порядка** и тип данных **комплексное число**. Прочитать из файла массив комплексных полиномов, запросить у пользователя комплексное число z, на экран вывести вывести на экран полином, значение которого на заданном z будет минимальным по модулю.

9. Вариант

Создать тип данных **студенты** (фамилия, имя, год рождения, номер группы). Записать набор таких данных в бинарный файл записей переменной длины в отсортированном по группам и фамилиям виде. Каждое текстовое поле начинается с байта, указывающего на длину этого поля, запись содержит дополнительный байт, вначале, задающий длину всей записи, весь файл в начале содержит целое число, задающее количество записей в нем.
Написать функцию, распечатывающую на экране всех студентов указанной пользователем группы на экране.

10. Вариант

Написать программу, подсчитывающую число уникальных слов в текстовом файле.
Подсчет вести в массиве указателей на структуры вида: **слово, частота**, отсортированном по частотам слов. Использовать функцию выделения динамической памяти.
Результирующую таблицу частот напечатать на экране.

11. Вариант

Создать тип данных **квадратные уравнения** ($ax^2+bx+c=0$), и тип данных **корни квадратного уравнения**. Написать функцию решения квадратных уравнений **`solve()`**, возвращающую решение заданного уравнения. Прочитать уравнения из файла, на экране напечатать решение каждого из них.

12. Вариант

Написать программу, создающую бинарный индексный файл к любому заданному текстовому файлу. **Индекс** реализовать в виде набора структур вида: слово, смещение в байтах от начала файла строки, содержащей это слово. По запросу пользователя вывести на экран целиком все строки исходного файла, содержащие указанное пользователем слово.

13. Вариант

Создать тип данных **комплексные числа**. Прочитать из файла массив комплексных чисел, отсортировать его с помощью библиотечной функции **`qsort()`**. Для использования функции **`qsort()`** создать свою функцию сравнения двух комплексных чисел по модулю.
Результат напечатать на экране специально созданной функцией.

14. Вариант

Создать тип данных многомерный **вектор чисел** с операциями вставки и удаления нового элемента в произвольное место вектора. Для изменения длины вектора использовать функции динамического выделения памяти. Для заданных в двух разных текстовых файлах векторов построить вектор, состоящий только из их общих элементов.
Результат вывести в третий файл.

15. Вариант

Создать тип данных **прямоугольник**, как набор **двух точек** (задаваемых собственным типом) – левой верхней и правой нижней. Для прямоугольников реализовать функции пересечения и объединения. Из двух файлов считать два массива прямоугольников, внутри каждого массива – объединить все прямоугольники, между массивами их пересечь. Результат вывести на экран.

16. Вариант

Создать тип данных **матрица** и тип данных **вектор**. Прочитать из одного файла матрицу неизвестной заранее размерности, из другого – вектор, также неизвестной размерности. Перемножить матрицу на вектор (дополняя его нулями или усекая по необходимости).
Результат напечатать на экране специально созданной функцией.

17. Вариант

Линейный **односвязный список** – это динамический список, каждый элемент которого состоит из двух полей. Одно поле содержит полезную информацию, другое поле содержит указатель на следующий элемент списка, например:

```
struct list { list* next; double elem; };
```

Прочитать из двух файлов два списка **географических объектов** формата: название, широта, долгота и пересечь их по координатам, считая координаты полностью совпадающими, если они различаются не более, чем на одну сотую градуса по широте и долготе. Из совпавших по координатам элементов в результирующий список записать любой (но один), результат вывести на экран.

18. Вариант

Создать тип данных **полиномы произвольного порядка**. Создать для них функции сложения, умножения на число, умножения самих полиномов и суперпозиции (когда один полином подставляется в другой в качестве аргумента, при этом должны использоваться операции сложения, умножения и умножения на число). Считать из файла два разных полинома, подставить второй из них аргументом в первый.
Результат напечатать на экране.

19. Вариант

Линейный **односвязный список** – это динамический список, каждый элемент которого состоит из двух полей. Одно поле содержит полезную информацию, другое поле содержит указатель на следующий элемент списка, например:

```
struct list { list* next; double elem; };
```

Составить **частотную таблицу** (тип данных: слово, частота) для уникальных слов в заданном текстовом файле, проводя линейный поиск новых слов в отсортированном по алфавиту списке. Найденным словам увеличивать счетчик частоты, не найденные – вставлять в нужную позицию списка. Результат вывести на экран.

20. Вариант

Создать тип данных **входные данные для решения уравнения** методом деления отрезка пополам (границы интервала, пороги точности и невязки) и тип данных ответ (корень, точность, невязка, количество итераций). Написать функцию поиска корня любого уравнения из семинара 2 с использованием этих структур.

21. Вариант

Создать тип данных **отсортированный вектор** (длина, указатель на массив элементов), написать для него функции добавления элемента (так, чтобы сохранить порядок сортировки) и удаления первого элемента. В качестве элементов вектора использовать указатель на уникальные слова из заданного текстового файла.
Список уникальных слов вывести на экран.

22. Вариант

Создать типы данных для **геометрических объектов** на плоскости: точка, отрезок (две точки), круг (точка цента и радиус), прямоугольник (два отрезка, задающие стороны), равносторонний треугольник (три отрезка, задающие стороны). Прочитать из файла набор геометрических объектов, вывести их звездочками в поле зрения 25 строк на 80 столбцов на экран.

23. Вариант

Линейный **односвязный список** – это динамический список, каждый элемент которого состоит из двух полей. Одно поле содержит полезную информацию, другое поле содержит указатель на следующий элемент списка, например:

```
struct list { list* next; double elem; };
```

Составить **частотную таблицу** (тип данных: слово, частота) для уникальных слов в заданном текстовом файле, проводя линейный поиск новых слов в отсортированном по частотам списке. Найденным словам увеличивать счетчик частоты, не найденные – вставлять в конец списка. Результат вывести на экран.

24. Вариант

Линейный **односвязный список** – это динамический список, каждый элемент которого состоит из двух полей. Одно поле содержит полезную информацию, другое поле содержит указатель на следующий элемент списка, например:

```
struct list { list* next; double elem; };
```

Создать тип данных **функция** (текстовое обозначение функции, указатель на соответствующую функцию), определить несколько элементарных функций (sin, cos, log и т.п.). Пользователь вводит простое арифметическое выражение (без учета приоритета операций) с указанными функциями, например:

```
sin * sin + cos * cos
```

что означает:

```
(sin(x) * sin(x) + cos(x)) * cos(x)
```

Нужно построить односвязный список, в котором элементы задают последовательно указанные функции и операции между ними, а также отдельную функцию, которая вычисляет все выражение, определенное этим списком для заданного пользователем значения x.
Можно воспользоваться типом union для того, чтобы в элементах списка можно было задавать или функцию или операцию. Результат напечатать на экране специально созданной функцией.

25. Вариант

В файле записана система линейных уравнений в виде матрицы коэффициентов, где последний столбец – это вектор правых частей уравнения. Прочитать этот файл в массив векторов-строк, решить систему методом Гаусса (приведением к треугольному виду).
Для решения этой системы нужно завести тип: вектор-строка и определить необходимые арифметические операции с этим типом. Результат вывести на экран.