1. Структуры

Структура — это совокупность переменных, объединенных под одним именем. С помощью структур удобно размещать связанные между собой элементы информации. Объявление структуры создает шаблон, который можно использовать для создания ее объектов (экземпляров этой структуры). Переменные, из которых состоит структура, называются элементами структуры или полями.

1.1.Простые структуры

Определение структуры

```
struct имя_структуры {
    поле 1;
    поле 2;
    поле 3;
    -------
    поле N;
};
```

Ключевое слово **`struct`** сообщает компилятору, что объявляется (декларируется) структура. Далее указывается имя структуры и в фигурных скобках поля, которые будет иметь данная структура. Поля структуры могут быть любого типа. Это могут быть как целые или дробные числа, числовые массивы, строки, массивы строк и даже другие структуры.

Как правило, поля структуры связаны друг с другом по смыслу. Например, информацию о человеке, состоящую из имени, адреса и телефона логично представить в виде структуры.

```
struct person {
    char name[250];
    int age;
    char address[250];
    char telephone[250];
};
```

Обратите внимание, что объявление завершается точкой с запятой, потому что объявление структуры является оператором. Кроме того, имя структуры `person` идентифицирует эту конкретную структуру данных и является спецификатором ее типа.

В данном случае на самом деле никакая переменная не создается. Всего лишь определяется вид данных. Когда вы объявляете структуру, то определяете агрегатный тип, а не переменную. И пока вы не объявите переменную этого типа, то существовать она не будет.
Чтобы объявить переменную (то есть физический объект) типа `person`, напишем следующее:

```
struct person client;
```

В этом операторе объявлена переменная типа `person`, которая называется `client`. Таким образом, `person` описывает вид структуры (ее тип), а `client` является экземпляром (объектом) этой структуры.
Когда объявляется переменная-структура, компилятор автоматически выделяет количество памяти, достаточное, чтобы разместить все ее поля.

Доступ к полям структуры

Доступ к отдельным полям структуры осуществляется с помощью оператора «`.`», который называют оператором точка или оператором доступа к полю структуры. Например, в следующем выражении полю `age` в уже объявленной переменной-структуре `client` присваивается значение `45`:

```
client.age = 45;
```

Это отдельное поле определяется именем объекта (в данном случае `client`), за которым следует точка, а затем именем самого этого члена (в данном случае `age`). В общем виде использование оператора точка для доступа к члену структуры выглядит таким образом:

```
имя_объекта.имя_поля
```

Таким образом, чтобы вывести на экран значение `age`, следует написать:

```
printf("%d", client.age);
```

Будет выведено значение, которое находится в поле `age` переменной-структуры `client`. Точно так же в вызове `gets()` можно использовать массив символов `client.name`:

```
gets(client.name);
```

Так как `name` является массивом символов, то чтобы получить доступ к отдельным символам в массиве `client.name`, можно использовать индексы вместе с `name`. Например, с помощью следующего кода можно посимвольно вывести на экран содержимое `client.name`:

```
for(i = 0; i < strlen(client.name); i++)
    putchar(client.name[i]);
```

Обратите внимание, что индексируется именно `name` (а не `client`). `client` — это имя всего объекта-структуры, а `name` — имя поля этой структуры. Таким образом, если требуется индексировать поле структуры, то индекс необходимо указывать после имени этого поля.

Присваивание структур

Информация, которая находится в одной структуре, может быть присвоена другой структуре того же типа при помощи единственного оператора присваивания. Нет необходимости присваивать значения каждого поля в отдельности.

```
//объявление структуры «мобильный телефон» с полями: фирма, модель, операционная система и цена
struct mobile_phone {
    char firma_name[20];
    char model_name[20];
    char OS_system[10];
    int price;
};

//объявляем две переменные-структуры
struct mobile_phone phone1, phone2;

//вводим с клавиатуры значение полей для phone1
gets(phone1.firma_name);
gets(phone1.model_name);
gets(phone1.OS_system);
scanf("%d", &phone1.price);

//копируем структуру phone1 в структуру phone2
phone2 = phone1;
```

```
    //теперь обе переменные-структуры имеют одинаковые значения всех полей.
```

Использование ключевого слова `typedef`

Это ключевое слово позволяет упростить работу с некоторыми типами данных. Используя **`typedef`**, можно определять новые имена типов данных. На самом деле таким способом новый тип данных не создается, а всего лишь определяется новое имя для уже существующего типа. Такие выражения могут помочь в самодокументировании кода, позволяя давать понятные имена стандартным типам данных. Общий вид декларации **`typedef`** (оператора **`typedef`**) такой:

```
typedef тип новое_имя;
```

- `тип` — это любой тип данных языка Си;
- `новое_имя` — это новое имя указанного типа.

Важно понимать, что новое имя является дополнением к уже существующему имени, а не его заменой.

Например, для **`float`** можно создать новое имя:

```
typedef float new_float;
```

Это выражение дает компилятору указание считать `new_float` еще одним именем **`float`**. Затем, используя `new_float`, можно создать переменную типа **`float`**:

```
new_float number;
```

У нас появилась дробная переменная `number` типа `new_float`, а `new_float` является еще одним именем типа **`float`**.

Теперь, когда имя `new_float` определено, его можно использовать и в другом операторе **`typedef`**. Например, выражение

```
typedef new_float another_new_float;
```

дает компилятору указание признавать `another_new_float` в качестве еще одного имени `new_float`, которое в свою очередь является еще одним именем **`float`**.

Использование операторов **typedef** может облегчить чтение кода и его перенос на новую машину. Однако следует помнить, что новый физический тип данных таким способом вы не создадите.

Ключевое слово **typedef** может быть использовано при объявлении структуры:

```
typedef struct {
  char firma_name[250];
  char model_name[250];
  char OS_system[250];
  int price;
} mobile_phone;
```

Таким образом, последнее слово `mobile_phone` — это новое имя типа. А тип описывается структурой:

```
struct {
  char firma_name[250];
  char model_name[250];
  char OS_system[250];
  int price;
}
```

Теперь мы можем использовать слово `mobile_phone` как обозначение описанной структуры. При этом объявление экземпляров структуры будет выглядеть следующим образом:

```
mobile_phone phone_list, phone_list2, my_phone;
```

Пример. Создать структуру `Функция` с полями: `x[N]`, `y[N]`, `z[N]`. Заполнить массивы координат x и y значениями:

- по `x`: от `-3` с шагом `2.5`;
- по `y`: от `15` с шагом `1.8`.

Вычислить значения поля `z[N]` по формуле: `z = 8x + 1.5y`.

```
#define N 40

//задаем структуру «функция»
typedef struct {
```

```
  double x[N];
  double y[N];
  double z[N];
} function;

int main(void) {

  int i;
  double h;
  function funk; //объявляем переменную-структуру


  //заполнение значениями поля double x[N]
  for(i = 0, h = -3; i < N; h += 2.5, i++)
    funk.x[i] = h;

  /заполнение значениями поля double y[N]
  for(i = 0, h = 15; i < N; h += 1.8, i++)
    funk.y[i] = h;

  //вычисление значений поля double z[N]
  for(i = 0; i < N; i++)
    funk.z[i] = 8 * funk.x[i] + 1.5 * funk.y[i];

  //вывод на экран значений всех полей
  for(i = 0; i < N; i++)
    printf("x = %7.2lf, y = %7.2lf, z = %7.2lf\n",
                              funk.x[i], funk.y[i], funk.z[i]);

  return EXIT_SUCCESS;
}
```

1.2. Массивы структур

Чтобы объявить массив структур, вначале необходимо определить структуру, а затем объявить переменную массива этого же типа. Например, чтобы объявить 100-элементный массив структур типа mobile_phone, нужно написать следующее:

```
#define N 100
typedef struct {
    char firma_name[20];
    char model_name[20];
```

```
    char OS_system[10];
    int price;
} mobile_phone;

mobile_phone phone_list[N];
```

Это выражение создаст 100 переменных, каждая из которых организована так, как определено в структуре mobile_phone.

Чтобы получить доступ к определенной структуре из массива структур, нужно указать имя массива структур с индексом. Например, чтобы вывести поле стоимость третьего элемента массива структур phone_list нужно сделать следующее:

```
printf("%d", phone_list[2].price);
```

В массивах структур индексирование так же начинается с 0.

Чтобы указать определенную структуру, находящуюся в массиве структур, необходимо указать имя этого массива с определенным индексом. А если нужно указать индекс определенного элемента в структуре, то необходимо указать индекс этого элемента. Таким образом, в результате выполнения следующего выражения первому символу поля model_name, находящегося в третьей структуре из phone_list, присваивается значение 'X'.

```
phone_list[2].model_name[0] = 'X';
```

Приведем пример работы с массивом структур, для которого полем является числовой или строковый массив. Структура «мама» состоит из полей: имя, список имен детей, массив, отражающий возраст детей. Рассмотрим три способа заполнения массива структур. Эти способы также подходят и для единичной структуры.

```
#define N 4 //количество мам
#define M 3 //количество детей у каждой мамы
typedef struct {
  char mum_name[20];
  char childrent_name[M][20];
  int childrent_age[M];
} mother;

//три способа заполнить структуру значениями:
```

```
    //первый способ: задать чему равны поля структур при объявлении
(инициализация при объявлении)

    mother mammy[N] = {
        {"Anna", {"Katrin","Piter", "Rosa"}, 9, 5, 12},
        {"Rita", {"Misha", "Liza", "Olga"}, 14, 15, 21},
        {"Mary", {"Egor", "Alex", "Mike"}, 20, 7, 15},
        {"Sveta", {"Katrin", "Marta", "Erin"}, 28, 25, 5}
    };

    //второй способ: задать чему равны поля структур с клавиатуры
    for (i = 0; i < N; i++) {
      gets(mammy[i].mum_name);
      for (j = 0; j < M; j++) {
        gets(mammy[i].childrent_name[j]);
        scanf("%d\n", &mammy[i].childrent_age[j]);
      }
    }

    //третий способ: задать чему равны поля структур из файла
    FILE * file;
    if ((file = fopen("mammy_list.txt","r")) == 0) {
      printf("error");
      exit(1);
    }
    for (i = 0; i < N; i++) {
      fscanf(file, "%s", mammy[i].mum_name);
      for (j = 0; j < M; j++) {
        fscanf(file, "%s %d", mammy[i].childrent_name[j],
                                        &mammy[i].childrent_age[j]);
      }
    }
    fclose(file);
```

Пример. Задать структуру, описывающую студента и его успеваемость. Создать массив из 25ти студентов. Вывести на экран таблицу со всеми полями получившегося массива структур.

```
    #define N 25
    //объявляем структуру «студент»
    typedef struct {
      char name[100];
      int age;
```

```
    int course;
    char group[20];
    double avr_mark;
  } student;

  int main(void) {
    int i = 0;

    //объявляем массив студентов из N элементов
    student muctr_stud[N];

    FILE * file;
    if ((file = fopen("student_list.txt","r")) == 0) {
      printf("error");
      exit(1);
    }
      //заполняем поля массива структур из файла
    for (i = 0; i < N; i++) {
      fscanf(file, "%s %d %d %s %lf", muctr_stud[i].name,
&muctr_stud[i].age, &muctr_stud[i].course, muctr_stud[i].group,
&muctr_stud[i].avr_mark);
    }
    fclose(file);


    printf("Таблица всех студентов\n");

    //выводим шапку таблицы
    printf("%10s %5s %5s %10s %15s\n\n","student_name", "age",
"course", "group", "average_mark");

    //выводим строчки таблицы
    for (i = 0; i < N; i++) {
        printf("%10s %5d %5d %10s %15.1lf\n", muctr_stud[i].name,
muctr_stud[i].age, muctr_stud[i].course, muctr_stud[i].group,
muctr_stud[i].avr_mark);
    }
    return EXIT_SUCCESS;
  }
```

Пример. Задать структуру, описывающую домашних питомцев. Создать массив из 10ти животных. Вывести на экран таблицу со всеми полями получившегося массива структур. Вывести на экран клички всех питомцев, которые старше среднего возраста всех питомцев.

```c
#define N 7
//объявляем структуру «питомец»
typedef struct {
  char type[50];
  char name[100];
  int age;
  char favourite_food [20];
}pet;

int main(void) {
  int i = 0;
  pet pet_list[N]; //объявляем массив из N питомцев

  FILE * file;
  if ((file = fopen("pets_list.txt","r")) == 0) {
    printf("error");
    exit(1);
  }
  for (i = 0; i < N; i++) {
    fscanf(file, "%10s %10s %5d %10s", pet_list[i].type,
pet_list[i].name, &pet_list[i].age,
pet_list[i].favourite_food);
  }
  fclose(file);

  printf("Таблица всех питомцев\n");

  //выводим на экран шапку таблицы
  printf("%10s %10s %5s %10s\n\n", "pet_type", "name", "age",
"favourite_food");
  //выводим на экран строчки таблицы
  for (i = 0; i < N; i++) {
    printf("%10s %10s %5d %10s", pet_list[i].type,
pet_list[i].name, pet_list[i].age, pet_list[i].favourite_food);
  }

  //суммируем возраст всех питомцев
  int sum_age = 0;
  for (i = 0; i < N; i++) {
    sum_age += pet_list[i].age;
}

  //расчитываем средний возраст
  double avr_age = (double)sum_age / N;
```

```
  printf("Средний возраст питомцев равен %lf", avr_age);

  //выводим на экран имена питомцев с возрастом больше среднего
  printf("Клички питомцев, чей возраст больше среднего\n\n");

  for (i = 0; i < N; i++) {
    if(pet_list[i].age > avr_age) {
      printf("%10s\n", pet_list[i].name);
    }
  }

  return EXIT_SUCCESS;
}
```

1.3. Вложенные структуры

Рассмотрим ситуацию, когда одним из полей структуры является другая структура. В этом случае сначало необходимо определить вложенную структуру. А затем использовать ее при определении основной структуры.

Рассмотрим пример, когда существует структура Мама с полями: Имя, Возраст, Массив детей; где Массив детей – это массив структур Ребенок с полями Имя, Пол, Возраст.

```
#define N 4 //количество мам
#define M 3 //количество детей у каждой мамы

typedef struct {
  char child_name[20];
  int child_age;
} child;

typedef struct {
  char mum_name[20];
  int mum_age;
  child child_list[M];
} mother;

int main(void) {
  mother mum_list[N];
  FILE * file, *file2;
  int i = 0, j = 0;
  file = fopen("mum_file", "r");
  file2 = fopen("child_file", "r");
  for(i = 0; i < N; i++) {
    fscanf(file, "%s %d", mum_list[i].mum_name,
                                  &mum_list[i].mum_age);
```

```
    for(j = 0; j < M; j++) {
      fscanf(file2, "%s %d",
                    mum_list[i].child_list[j].child_name,
                       &mum_list[i].child_list[j].child_age);
    }
  }
  fclose(file);
  fclose(file2);
  return EXIT_SUCCESS;
}
```

В случае, если у каждой из мам разное количество детей, то при определении структуры невозможно указать размер массива структур типа `child`. В таком случае необходимо использовать динамическую память для работы с массивом типа `child`.

```
#define N 4 //количество мам

typedef struct {
  char child_name[20];
  int child_age;
} child;

typedef struct {
  char mum_name[20];
  int mum_age;
  int child_num;
  child * child_list; //указатель на массив детей
} mother;

int main(void) {
  mother mum_list[N];
  FILE * file, *file2;
  int i = 0, j = 0;
  file = fopen("mum_file", "r");

//считываем Имя, Возраст и Количество детей у каждой мамы
  for(i = 0; i < N; i++) {
    fscanf(file, "%s %d %d", mum_list[i].mum_name,
           &mum_list[i].mum_age, &mum_list[i].child_num);
}
  fclose(file);

//Выделяем динамическую память для массива детей
  for(i = 0; i < N; i++) {
    mum_list[i].child_list =
       (child*)malloc(mum_list[i].child_num * sizeof(child));
  }

  file2 = fopen("child_file", "r");

//второй цикл от нуля до количества детей у каждой мамы
```

```
  for(i = 0; i < N; i++) {
    for(j = 0; j < mum_list[i].child_num; j++) {
      fscanf(file2, "%s %d",
                  mum_list[i].child_list[j].child_name,
                     &mum_list[i].child_list[j].child_age);
    }
  }
  fclose(file2);

//освобождаем память
  for(i = 0; i < N; i++) {
    free(mum_list[i].child_list);
  }

  return EXIT_SUCCESS;
}
```