Перечисления.
Указатели на структуры.

Перечисляемый тип

В Си выделен отдельный тип «перечисление» (**enum**), задающий набор всех возможных **целочисленных** значений переменной этого типа. Обычно перечисления используются в качестве набора именованных констант (вместо **const int** или директив препроцессора **#define**).

Например:

```
enum boolean {F, T};
printf("%d\n", T);
```

При таком объявлении первое значение типа **boolean** имеет величину 0, второе – 1 и т. д. (пример lect-11-01.c).

Объявление: **enum** <имя> {<названия и значения>}.

Перечисляемый тип

Применение типа «перечисление» позволяет получить фиксированный набор именованных целочисленных констант. Повышается наглядность программы и устраняются директивы **`#define`**.

1. Перечисления могут быть анонимными (пример lect-11-02.c)

```
enum {F, T};
printf("%d\n", T);
```

2. Можно описать переменную типа **`enum`** и присваивать ей значения из набора.

Перечисляемый тип

3. Память выделяется для одного `int`, а не для нескольких (пример lect-11-03.c).

```
enum colors {
                Red     = 0xff0000,
                Green   = 0x00ff00,
                Blue    = 0x0000ff,
                Black   = 0x0,
                White   = 0xffffff
            };
enum colors oneColor;

oneColor=Blue;
printf("%d\n", oneColor);
printf("%ld\n", sizeof(enum colors));
```

Перечисляемый тип

4. Можно работать как с числовыми значениями, так и с названиями элементов «перечисления».

5. Можно установить начальное значение элементов «перечисления» (пример lect-11-04.c).

```
enum lg0 {c, cpp, pascal, ada};
enum lg1 {python=11, perl, php, js};
enum lg0 x;
enum lg1 y;
int z;
x=cpp;
printf("Current language= %d\n", x);
z=x+123;
printf("Current value= %d\n", z);
if(x==cpp) printf("Current language is cpp\n");
y=perl;
printf("Value of y: %d\n", y);
```

Константа, значение которой не указано, будет иметь значение на 1 больше, чем предыдущая.

Перечисляемый тип

В значениях констант «перечисления» можно указать любую операцию, результат которой может вычислен компилятором:

```
enum example {
              first = 10;
              second = first*20;
              third = first+1000;
              last = third;
            };
```

Можно объявлять переменные сразу после описания типа (как для структур):

```
enum state {
            running, blocked, ready
           } new_state, last_state;
```

Динамические массивы структур

В реальных задачах количество элементов массива структур предсказать нельзя.

При работе с файлами CSV можно посчитать количество строк, потом выделить память (пример lect-11-05.c).

```
n=0;
while((fgets(s1,maxlen,df))!=NULL) n++;
rewind(df);
…
arr=(struct sss*)malloc(n*sizeof(struct sss));
```

Если такой возможности нет (например, при вводе с клавиатуры), то приходится использовать **`realloc()`**. При этом должен быть определен признак окончания работы.

При вводе с клавиатуры можно запрашивать подтверждение на продолжение ввода (см. также пример lect-11-06.c).

Динамические массивы структур

1. При динамическом выделении памяти требуется проверять успешность выделения.

2. Динамически выделенная память должна быть очищена, при выделении памяти не для всех элементов массива нужно очистить все что было выделено.

3. Если при очередном **`realloc()`** адрес массива стал **`NULL`**, то очищать уже нечего.

Указатели на структуры

При передаче структур в функции по значению необходимы существенные ресурсы:

если в структуре имеется большое количество элементов (полей) или некоторые элементы (поля) сами являются массивами, то при передаче структур функциям все это размещается в стеке (а у стека размер ограничен) + разбор и копирование структуры тоже требует времени.

Для более эффективного использования ресурсов лучше передавать не саму структуру, а указатель на нее (передача адреса параметра).

В языке Си указатели на структуры определены, как и указатели на любой другой вид объектов.

Указатели на структуры

Как и другие указатели, указатель на структуру объявляется с помощью звездочки `*`, которую помещают перед именем переменной-структуры (пример lect-11-07.c).

```
typedef struct student studs;
…
studs *stud0=NULL;
…
stud0=(studs*)malloc(sizeof(studs));
```

Для структуры нужно выделить память, после окончания работы — очистить выделенную память (**`free(stud0)`**).

При очистке памяти есть опасность получить ошибку времени выполнения (**`double free`**).

Указатели на структуры

1. Указатель на структуру позволяет передать структуру в функцию с помощью вызова по адресу.

2. Когда функции передается только адрес структуры, то вызовы функции выполняются быстрее, чем при передаче структуры целиком (не нужно разбирать структуру).

3. Передача указателя позволяет функции модифицировать содержимое структуры, указанной в качестве фактического параметра.

Синтаксис обращения к полям:

либо `(*stud0).name`

либо `stud0→name` (`*stud.name` — указатель на поле).

Указатели на структуры

С использованием указателя относительно просто вынести заполнение полей структуры в функцию (пример lect-11-08.c).

```
studs *struct_fill(char **str) /* на входе – массив строк */
{
    studs *str0=NULL;

    str0=(studs*)malloc(sizeof(studs));
    if(str0!=NULL)
    {
        /* заполняем поля из элементов массива строк */
    }
    return str0;
}
```

Для заполнения полей, определенных как строки в динамической памяти, используются присваивания (т. к. работа идет с адресами).

Динамический массив указателей на структуры

Объявляется аналогично динамическому массиву строк.

Технология работы та же самая (пример lect-11-09.c).

```
typedef struct student studs;
…
studs **stud0=NULL;
…
stud0=(studs**)malloc(n*sizeof(studs*)); /* это массив!! */
…
stud0[i]=struct_fill(s2); /* заполнение, s2 – массив строк */
```

Поля структуры не индексируются! Нужно знать имена и типы полей!

Можно ли перемещаться по полям структуры с использованием указателей и адресной арифметики?

Динамический массив указателей на структуры

Одна из типовых задач — найти в массиве структур элементы с заданным значением поля (с заданными значениями полей).

См. пример lect-11-10.c (Типы — те же).

Функция поиска (по полю gender):

```
studs *ssearch(studs *str0, char gender)
{
    studs *found;
    found=(studs*)malloc(sizeof(studs));
    if(found!=NULL)
    {
        if((str0->gender)==gender) found=str0;
        else found=NULL;
    }
    return found;
}
```

Динамический массив указателей на структуры

Применение в коде:

```
scanf("%c", &gender);
…
stud0[i]=struct_fill(s2); /* элемент массива структур */
                          /* из массива строк */
f=ssearch(stud0[i],gender); /* имеет тип studs* */
if(f!=NULL)
   {
      /* вывод структуры */
   }
```

Динамический массив указателей на структуры

Еще одна из типовых задач — сортировка массива структур по заданному полю (сортировка по среднему баллу – пример lect-11-11.c).

```
void sort_rating(studs **str0, int n) /*n – кол-во элементов*/
{
    studs *tmp_struct;
    int i,j;
    for(i=0;i<n;i=i+1)
    {
        tmp_struct=str0[i];
   for(j=i-1;(j>=0)&&((str0[j]→average) >
        (tmp_struct->average));j--)
        str0[j+1]=str0[j];
        str0[j+1]=tmp_struct;
     }
}
```

Сортировка вставками, используем возможность присваивания структур.