Содержание

структуры

Объявление и определение структуры

Объявление структуры:
`struct имя_структуры;`

Объявленна, но неопределенная, структура считаеться неполным типом и сделать с ней что либо нельзя тк ее размер без определения неизвестен. Объявить структуру можно несколько раз, а определить лишь раз.

Использование структуры

```
// определение структуры person
struct person {
  char * name;
  int age;
};

int main(void) {
  // определение переменной, которая представляет структуру person
  struct person tom;
}
```

определение структуры и ее переменной сразу

```
struct person{
  int age;
  char * name;
} tom;
```

при таком обявлении переменной можно не указывать имя структуры.

определение структуры через typedef (тн псевдоним структуры):

```
#include <stdio.h>

typedef struct {
  int age;
  char* name;
} person;

int main(void) {
  person tom = {23, "Tom"};
  printf("Name:%s \t Age: %d\n", tom.name, tom.age);
  return 0;
```

```
}
```

Инициализация структуры

По позиции: значения передаются элементам структуры в том порядке, в котором они следуют в структуре:

```
struct person tom = {23, "Tom"};
```

Так как в структуре person первым определено свойство age, которое представляет тип int - число, то в фигурных скобках вначале идет число, которое передается элементу age. Вторым идет элемент name, который представляет указатель на тип char или строку, соответственно вторым идет строка. И так далее для всех элементов структуры по порядку.

По имени: значения передаются элементам структуры по имени, независимо от порядка:

```
struct person tom = {.name="Tom", .age=23};
```

В этом случае перед именем элемента указывается точка, например, .name.

инициализация переменной при объявлении структуры:

```
struct person {
  int age;
  char * name;
} tom = {38, "Tom"};
```

Обращение к элементам структуры

имя_переменной_структуры.имя_элемента

```
#include <stdio.h>

struct person {
  int age;
  char *name;
};

int main(void) {
  struct person tom = {23, "Tom"};
  printf("Age: %d \t Name: %s", tom.age, tom.name);
  return 0;
}
```

Составные литералы

Составные литералы (compound literals) позволяют сократить код и использовать литералы, которые представляют структуру, без определения переменной структуры

```c
#include <stdio.h>

typedef struct {
  int x;
  int y;
} Point;

void print_point(Point p) {
  printf("(X:%d; Y:%d)\n", p.x, p.y);
}

Point create_point(int x, int y) {
  return (Point) {x, y}; // возвращаем строковой литерал
}
int main() {
  Point p = create_point(170, 150);
  print_point(p);
  return 0;
}
```