Структуры (введение). Линейный односвязный список.

Структуры

Структура представляет собой одну или несколько переменных (возможно разного типа), которые объединены под одним именем.

Структуры помогают в организации сложных данных, потому что позволяют описывать множество логически связанных между собой отдельных элементов как единое целое.

Структуры

Синтаксис объявления структуры

```
struct имя
{
    тип_1 имя_1;
    тип_2 имя_2;
    ...
    тип_N имя_N;
}; // точка с запятой обязательна!
```

Переменные, расположенные в объявлении структуры, называются ее *полями (элементами, членами)*.

Поля структуры могут иметь любой тип, кроме типа той же структуры.

Структуры

```
struct date
{
    int day;
    int month;
    int year;
};

struct person
{
    char        name[32];
    struct date birthday;
};

// Структуры и массивы могут комбинироваться без каких-либо
// ограничений.
```

```
struct point_2d
{
   double x, y;
   // Если поля имеют один тип,
   // их можно перечислить через
   // запятую.
};

struct point_2d triangle[3];
```

Инициализация структур

```
struct date
{
    int day;
    int month;
    int year;
};
struct person
{
    char         name[32];
    struct date birthday;
};
int main(void)
{
    struct date today =
              {19, 9, 2016};
```

```
struct date day = {19};

struct date year = {, , 2016};
// error: expected expression before ',' token

struct person rector =
    {"Aleksandrov", {7, 4, 1951}};

struct date holidays[] =
{
    {4, 11, 2016},
    {5, 11, 2016},
    {6, 11, 2016}
};
```

Операции над структурами

Доступ к полю структуры осуществляется с помощью операции “.”, а если доступ к самой структуре осуществляется по указателю, то с помощью операции “->”.

Операция	Название	Нотация	Класс	Приоритет	Ассоциат.
.<имя>	Прямой выбор поля <имя>	**X.<имя>**	Постфиксные	16	Слева направо
-><имя>	Выбор поля <имя> через указатель	**X-><имя>**			

Операции над структурами

```
struct date today, *tomorrow, some_date;

today.day   = 19;
today.month = 9;
today.year  = 2016;

tomorrow = malloc(sizeof(struct date));
if (!tomorrow)
    ...

(*tomorrow).day = 20;
tomorrow->month = 9;
tomorrow->year  = 2016;
```

Операции над структурами

Структурные переменные одного типа можно присваивать друг другу.

```
some_date = today;
```

Структуры нельзя сравнивать с помощью “==” и “!=”.

```
if (today == *tomorrow)
// error: invalid operands to binary == (have 'struct date'
// and 'struct date')
```

Структуры могут передаваться в функцию как параметры и возвращаться из функции в качестве ее значения.

Операции над структурами

```
void print(struct date d)
{
    printf("%02d.%02d.%04d", d.day, d.month, d.year);
}

void print_ex(const struct date *d)
{
    printf("%02d.%02d.%04d", d->day, d->month, d->year);
}
```

Передача структур с помощью указателей

- Эффективность.
- Необходимость изменения переменной (например, FILE*).

```
struct date get_student_date(void)
{
    struct date d = {25, 1, 2017};

    return d;
}
```

Достоинства и недостатки массивов

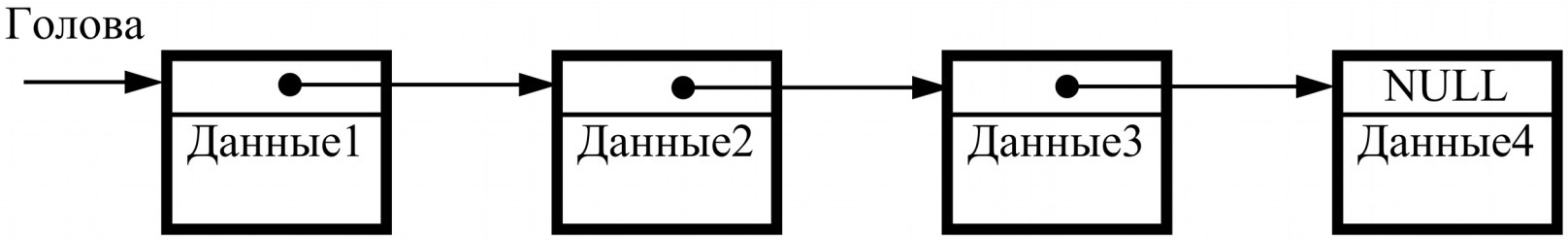
«+»

- Простота использования.
- Константное время доступа к любому элементу.
- Не тратят лишние ресурсы.
- Хорошо сочетаются с двоичным поиском.

«-»

- Хранение меняющегося набора значений.

Линейный односвязный список



Отличия списка от массива

- Размер массива фиксирован, а списка нет.
- Списки можно переформировывать, изменяя несколько указателей.
- При удалении или вставки нового элемента в список адрес остальных не меняется.

Элемент списка

```
struct list_node
{
    int num;

    struct list_node *next;
};

struct list_node* create_node(int num)
{
    struct list_node *node = malloc(sizeof(struct list_node));
    if (node)
    {
        node->num  = num;
        node->next = NULL;
    }
    return node;
}
```

Добавление элемента в список

```
struct list_node* add_front(struct list_node *head,
                            struct list_node *node)
{
    node->next = head;

    return node;
}
```

NB: функции, изменяющие список, должны возвращать указатель на новый первый элемент.

```
head = add_front(head, node);
```

Добавление элемента в список

```
struct list_node* add_end(struct list_node *head,
                                                struct list_node *node)
{
    struct list_node *cur = head;

    if (!head)
        return node;

    for ( ; cur->next; cur = cur->next)
        ;

    cur->next = node;

    return head;
}
```

Добавление элемента в конец нашего простого списка – операция порядка O(N). Чтобы добиться времени O(1), можно завести отдельный указатель на конец списка.

Поиск элемента в списке

```
struct list_node* lookup(struct list_node *head, int num)
{
    for ( ; head; head = head->next)
        if (head->num == num)
            return head;

    return NULL;
}
```

Поиск занимает время порядка O(N) и эту оценку не улучшить.

Обработка всех элементов списка

```
void print(struct list_node *head)
{
    printf("List:\n");
    for ( ; head; head = head->next)
        printf("%d ", head->num);

    printf("\n");
}
```

Освобождение списка

```
void free_all(struct list_node *head)
{
    struct list_node *next;

    for ( ; head; head = next)
    {
        next = head->next;
        free(head);
    }
}

// ОШИБКА
for ( ; head; head = head->next)
    free(head);
```

Удаление элемента по значению

```
struct list_node* del_by_value(struct list_node *head, int val)
{
    struct list_node *cur, *prev = NULL;
    for (cur = head; cur; cur = cur->next)
    {
        if (cur->num == val)
        {
            if (prev)
                prev->next = cur->next;
            else
                head = cur->next;
            free(cur);
            return  head;
        }
        prev = cur;
    }
    return NULL;
}
```