Абстрактные типы данных.
Списки: линейные односвязные.

СПбГЭТУ «ЛЭТИ»
ПЕРВЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ

Понятие абстрактного типа данных

Абстрактный тип данных (АТД) — это тип данных, для которого неизвестна внутренняя организация (устройство), а известен только набор операций с этим типом данных.

Примеры:

- Тип `FILE*`: есть функции (**`fopen()`**, **`fclose()`**, **`fputs()`**, **`fwrite()`** и т. п.), конкретное устройство знать не надо.
- Типы с плавающей точкой (**`float`**, **`double`**, **`long double`**) — операции с ними не зависят от конкретного типа.

Динамические информационные структуры

Переменные требуется объявлять для обеспечения выделения памяти для каждого типа данных.

Размер памяти → **`sizeof(<тип>)`**

Для каждого типа данных размер памяти неизменный.

Существуют задачи, требующие изменения размера памяти, выделяемой для объекта какого-то типа.

Можно использовать в таких задачах массивы, но это требует значительных затрат ресурсов в задачах с большим объемом данных:

Две операции над массивами - удаление и/или добавление элементов достаточно трудоемки.

Фактически при выполнении этих операций надо переписывать большие объемы информации из одной области памяти в другую.

Динамические информационные структуры

Для информации, которая в процессе работы программы изменяется в размерах, используются так называемые **«динамические структуры данных (ДСД)»**, они же **«динамические информационные структуры (ДИС)»** – структуры, которые формируются в процессе выполнения программы. Для этих структур операции «удалить и/или добавить элемент» не связаны с глобальной передислокацией данных.

Типы полей в структурах (еще раз)

- Символы (**`char`**)
- Числа (**`short, int, long, float, double …`**)
- Массивы (одномерные и многомерные)
- Строки
- Структуры
- Указатели на все перечисленное

+ указатели на функции (не имеет особого смысла в «классическом» Си)

Полем структуры может быть структура, но другого типа.

Полем структуры может быть указатель на структуру того же самого типа.

Списки

Список — совокупность объектов (элементов), в которой каждый объект содержит информацию о размещении связанного с ним объекта (объектов) – динамическая информационная структура.

При хранении списка в памяти размещение объектов определяется их адресами.

Операции с элементами списка не зависят от внутренней структуры элементов, поэтому список можно рассматривать как ***абстрактный тип данных*** (АТД).

Простейший вариант элемента списка — структура, содержащая одно информационное поле и указатель на такую же структуру.

element

Существует несколько вариантов (видов) списков.

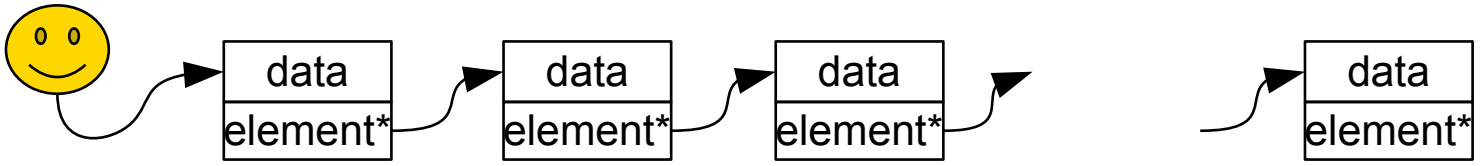
Линейный однонаправленный (односвязный) список

Для любого списка должна быть точка начала (указатель на первый элемент списка — «**голова**»).

Если этот указатель NULL — список пустой.

Если движение по списку (по указателям) возможно только в одном направлении, получается односвязный (однонаправленный) список.

Если существует последний элемент списка — список линейный («последний элемент» означает, что следующим является NULL).



L1-список

Линейный однонаправленный (односвязный) список

1. Определение структуры

```
struct LNode
{
   char data[32];
   struct LNode *next;
};

typedef struct LNode LN;
```

2. Определение «головы»

```
LN *LHead;
```

3. Инициализация «головы»

```
LHead=NULL;
```

Линейный однонаправленный (односвязный) список

4. Инициализация элемента списка

- Выделить память
- Установить значение поля данных
- Установить указатель next в NULL

5. «Привязка головы» к первому элементу списка

См. пример lect-13-01.c

Инициализации лучше делать в виде функций.

Линейный однонаправленный (односвязный) список

```
int main()
{
    LN *LHead=NULL; /* define and init head */
    LN *p=NULL;

    p=(LN*)malloc(sizeof(LN));
    if(p!=NULL)
    {
        strcpy(p->data,"dataword"); /* init node */
        p->next=NULL;
        LHead=p; /* make link head to node */

        printf("%p\n", LHead);
        free(p);
    }
    return 0;
}
```

Линейный однонаправленный (односвязный) список

«Голова» в виде указателя только сохраняет адрес первого элемента списка.

Можно создать «голову» в виде структуры, хранящей адреса первого и последнего элемента, а также имеющей информационные поля.

См. пример lect-13-02.c

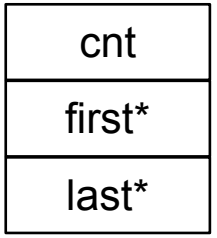
Линейный однонаправленный (односвязный) список

struct LNode

- id (порядковый номер элемента)
- data (область данных элемента)
- *next (указатель на следующий элемент)

struct LHead

- cnt (счетчик элементов)
- *first (указатель на первый элемент)
- *last (указатель на последний элемент)



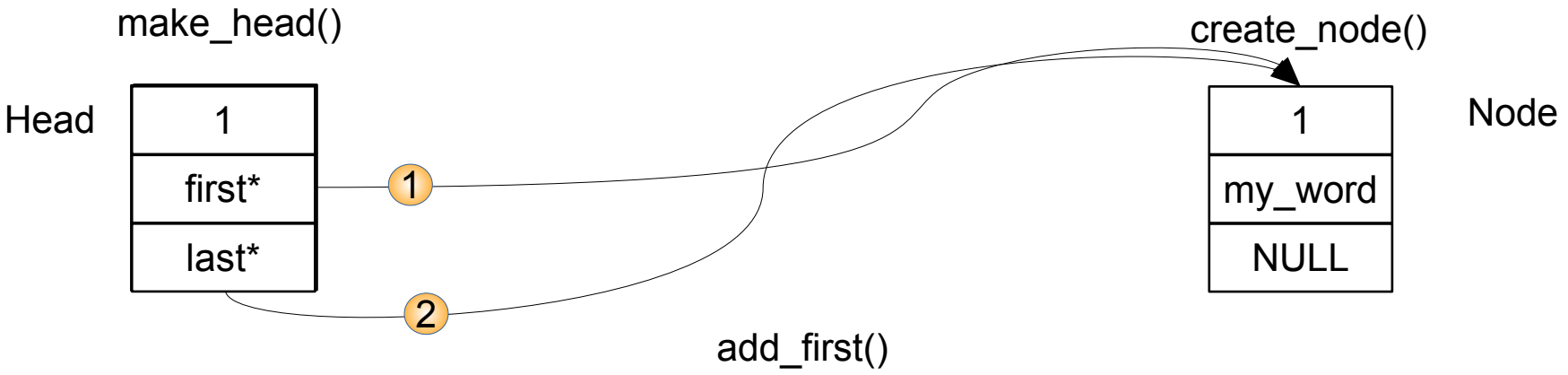
Линейный однонаправленный (односвязный) список

```
typedef struct LHead Head; /* datatype for head */
typedef struct LNode Node; /* datatype for node */

Head *make_head() /* head initialization */
{
    Head *ph=NULL;
    ph=(Head*)malloc(sizeof(Head));
    if(ph!=NULL)
    {
        ph->cnt=0;
        ph->first=NULL;
        ph->last=NULL;
    }
    return ph;
}
```

Линейный однонаправленный (односвязный) список

```
/* node creation and initialization */
Node *create_node(char *new_word, int slen)
{
   Node *new_node=NULL; /* pointer to new node */
   char *someword=NULL;
   new_node = (Node*)malloc(sizeof(Node));
   someword=(char*)malloc((slen+1)*sizeof(char));
   if(new_node&&someword)
   {
        new_node->id = 1;          /* setting node ID to 1 */
        strcpy(someword,new_word);
        new_node->word=someword;
        new_node->next = NULL;     /* next node is absent */
   }
   return new_node;  /* return of node address */
}
```

Линейный однонаправленный (односвязный) список



Линейный однонаправленный (односвязный) список

```
/* link head to first node */
void add_first(Head *my_head, Node *new_node)
{
    if(my_head&&new_node)
    {
        my_head->first = new_node;
        my_head->last = new_node;
        my_head->cnt++;
    }
}
```

Линейный однонаправленный (односвязный) список

```
int main()
{
    Head *p0=NULL;
    Node *p=NULL;
    … /* other definitions */
    p0=make_head();
    /* getting my_word */
    p=create_node(my_word, length);

    add_first(p0,p);
    printf("%d %s\n",p->id, p->word); /* results */
    printf("Initial Head: %p %d\n",p0->first, p0->cnt);
    if(p) free(p);
    if(p0) free(p0);
    return 0;
}
```

Типы информационных полей элемента списка

Элемент списка — это структура.
Типы информационных полей элемента списка такие же, как типы полей структуры.

- Символы (**`char`**)
- Целые числа (**`short, int, long`**)
- Вещественные числа (**`float, double`**)
- Массивы указанных выше типов
- Структуры
- Указатели на указанные выше типы.

Типы информационных полей элемента списка

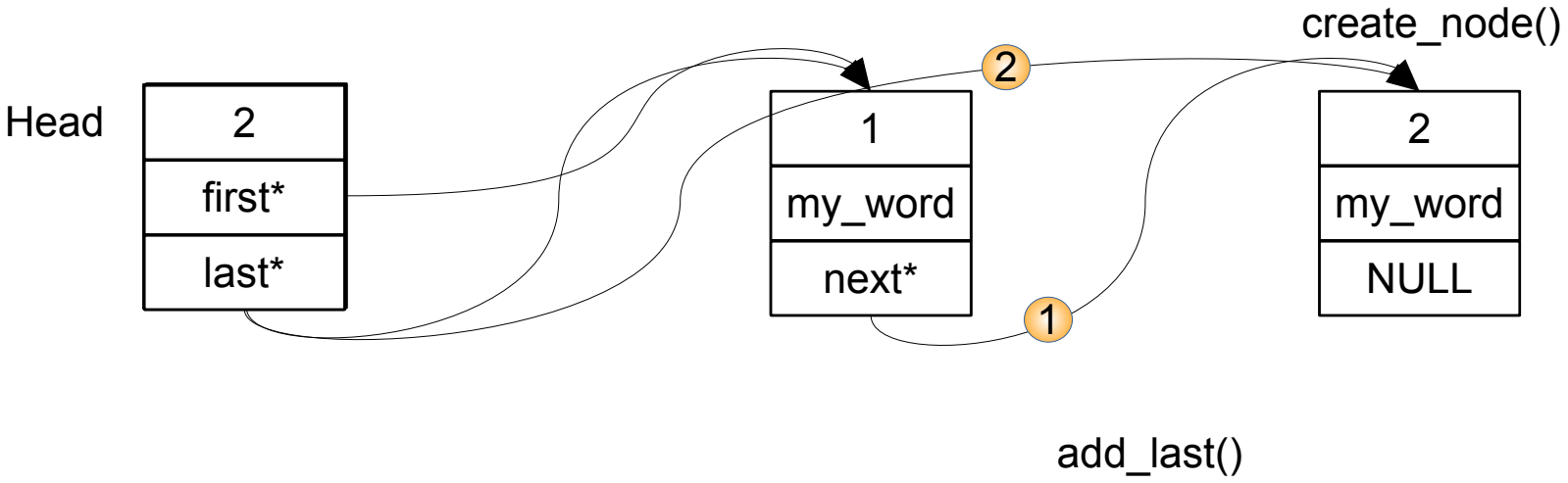
Пример:

```
struct bib_rec{
    char author[21];
    char title[128];
    int year;
    float price;
};

typedef struct bib_rec book;

struct book_elem{
    book info;
    book_elem *next;
};

typedef struct book_elem book_list;

book_list *BHead=NULL;
```

Операции с линейным односвязным списком

- Добавление элемента списка
- Определение длины списка
- Вывод значений информационный полей
- Поиск элемента по значению информационного поля
- Перестановка элементов списка
- Сортировка по значению информационного поля
- Удаление элемента/освобождение памяти.

Линейный однонаправленный (односвязный) список



см. пример lect-13-03.c

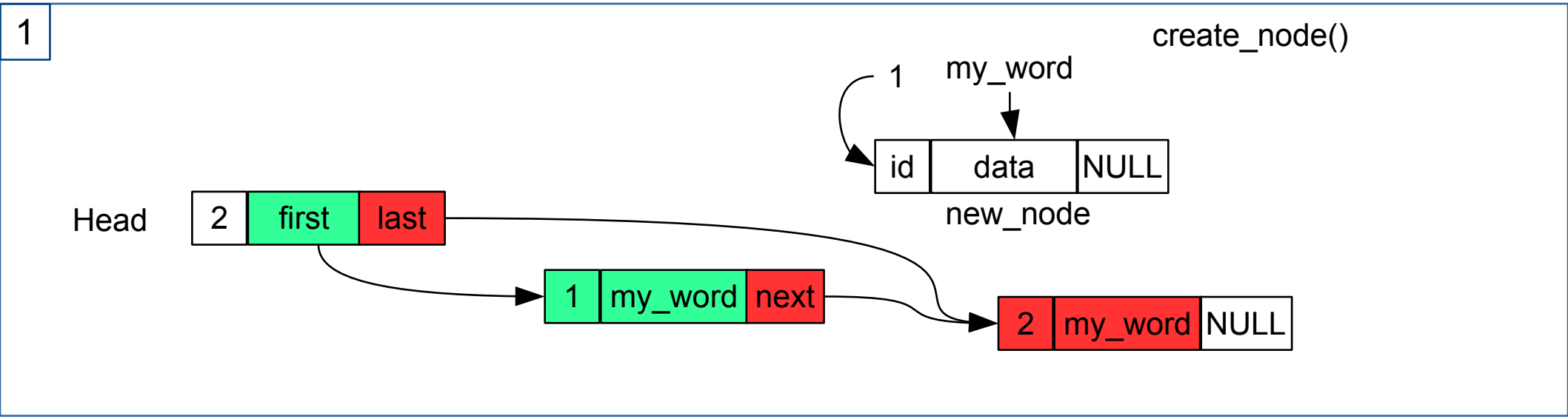
Добавление последнего элемента списка

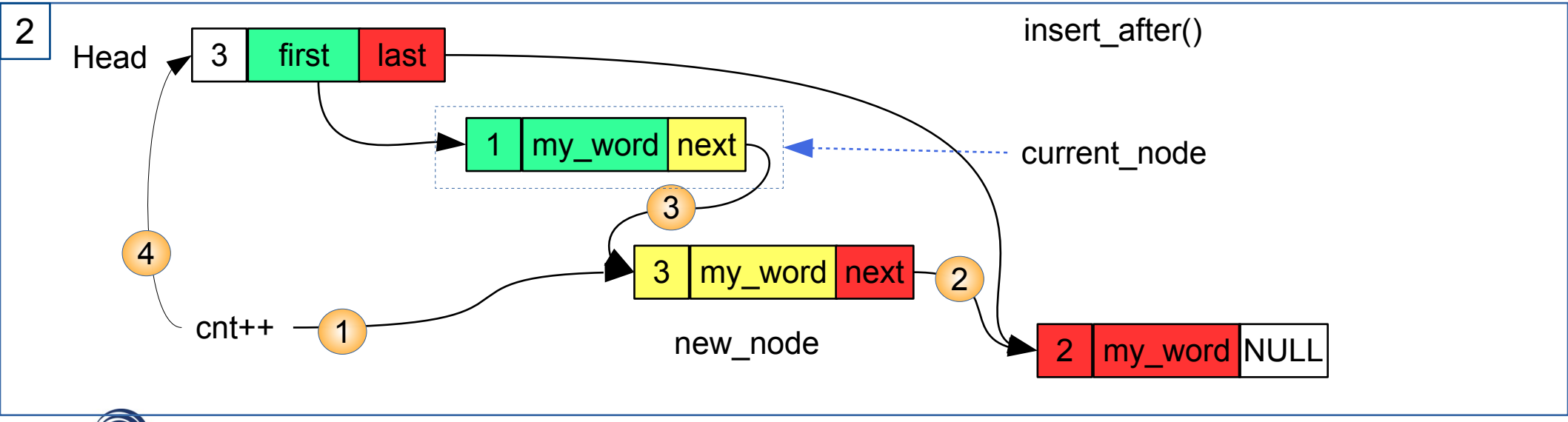
```
void add_last(Head *my_head, Node *new_node, Node *prev_node)
{
    int n;

    if(my_head && new_node && prev_node)
    {
        n=my_head->cnt+1;
        prev_node->next=new_node;
        new_node->id=n;
        my_head->last=new_node;
        my_head->cnt=n;
    }
}
```

Добавление последнего элемента списка

```
int main()
{
    Head *p0=NULL;
    Node *p=NULL,*p1=NULL;
    /* other defifnintions */
    p0=make_head();
    /* getting first word */
    p1=create_node(my_word, length);/*saving node address*/
    add_first(p0,p1);
    /* getting last word   */
    p=create_node(my_word, length);
    add_last(p0,p,p1);
    if(p) free(p);
    if(p1) free(p1);
    if(p0) free(p0);
    return 0;
}
```

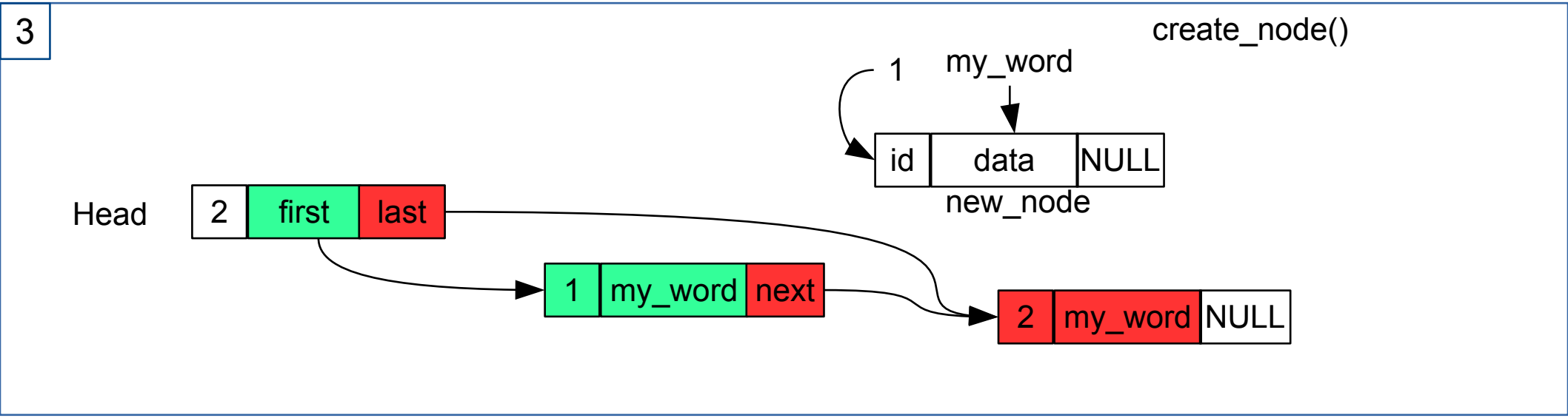
Вставка элемента списка – 1

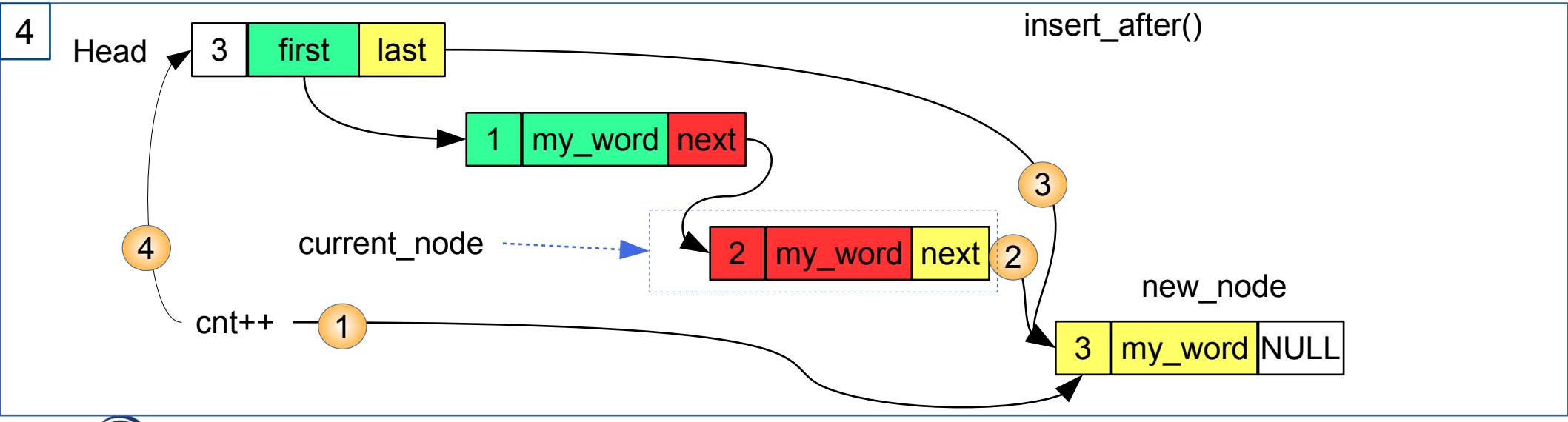




см. пример lect-13-04.c

Вставка элемента списка – 2





см. пример lect-13-04.c

Вставка элемента списка

```
void insert_after(Head *my_head, Node *new_node, Node *current_node)
{
    int n;
    if(my_head && new_node && current_node)
    {
        n=my_head->cnt+1;
        if(current_node->next==NULL)
        {
            current_node->next=new_node;
            my_head->last=new_node;
        }
        else
        {
            new_node->next = current_node->next;
            current_node->next=new_node;
        }
        new_node->id=n;
        my_head->cnt=n;
    }}
```