

Строки/структуры и динамическое выделение памяти

(часть 1)

Строки и динамическая память

```
// Тут нужны include-ы

#define NAME "Bauman Moscow State Technical University"

int main(void)
{
    char *name = malloc((strlen(NAME) + 1) * sizeof(char));

    if (name)
    {
        strcpy(name, NAME);
        printf("%s\n", name);
        free(name);
    }
    else
        printf("Cant allocate memory\n");

    return 0;
}
```

Строки и динамическая память

```
// Тут нужны include-ы
// Для компиляции -std=gnu99

#define NAME "Bauman Moscow State Technical University"

int main(void)
{
    char *name = strdup(NAME);    // string.h, POSIX (+ strdup)

    if (name)
    {
        printf("%s\n", name);
        free(name);
    }
    else
        printf("Cant allocate memory\n");

    return 0;
}
```

Строки и динамическая память

```
FILE *f;
char *line = NULL;
size_t len = 0;
ssize_t read;

// ...

f = fopen(argv[1], "r");
if (f)
{
    while ((read = getline(&line, &len, f)) != -1)
    {
        printf("len %d, read %d\n", (int) len, (int) read);
        printf("%s", line);
    }

    free(line);
    fclose(f);
}
```

Строки и динамическая память

```
#include <stdio.h>
```

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);    // POSIX
```

lineptr - либо NULL (и тогда в n - 0), либо указатель на буфер, выделенный с помощью malloc (и тогда в n - размер буфера). Если буфера не хватает, он будет перевыделен.

```
// C glibc 2.10
```

```
#define _POSIX_C_SOURCE 200809L
```

```
// До glibc 2.10
```

```
#define _GNU_SOURCE
```

Строки и динамическая память

```
int n, m;

n = snprintf(NULL, 0, "My name is %s. I live in %s.", NAME, CITY);
if (n > 0)
{
    char *line = malloc((n + 1) * sizeof(char));
    if (line)
    {
        m = snprintf(line, n + 1, "My name is %s. I live in %s.", NAME, CITY);

        printf("n = %d, m = %d\n", n, m);
        printf("%s\n", line);

        free(line);
    }
}
```

Строки и динамическая память

```
#define _GNU_SOURCE
#include <stdio.h>

// ...
{
    char *line = NULL;
    int n;

    n = asprintf(&line, "My name is %s. I live in %s.", NAME, CITY);
    if (n > 0)
    {
        printf("n = %d\n", n);
        printf("%s\n", line);

        free(line);
    }
}
```

Структуры с полями-указателями

В Си определена операция присваивания для структурных переменных одного типа. Эта операция фактически эквивалента копированию области памяти, занимаемой одной переменной, в область памяти, которую занимает другая.

При этом реализуется стратегия так называемого «*поверхностного копирования*» (англ., *shallow coping*), при котором копируется содержимое структурной переменной, но не копируется то, на что могут ссылаться поля структуры.

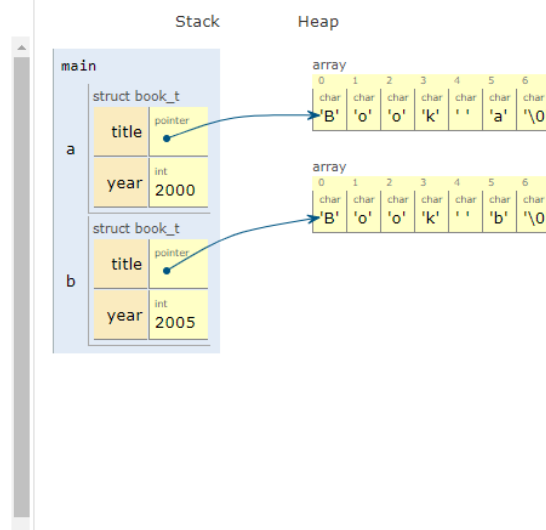
Структуры с полями-указателями

Иногда стратегия «поверхностного копирования» может приводить к ошибкам.

До присваивания

```
C (gcc 4.8, C11)
(known limitations)

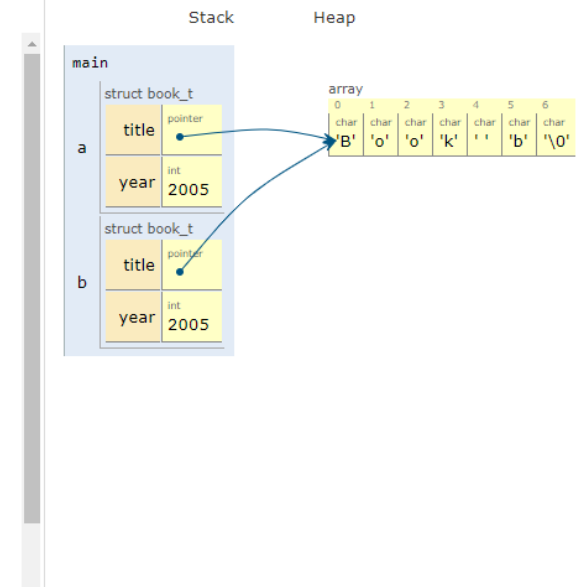
1 #include <string.h>
2 #include <stdlib.h>
3
4 int main(void)
5 {
6     struct book_t
7     {
8         char *title;
9         int year;
10    } a = { 0 }, b = { 0 };
11
12    a.title = strdup("Book a");
13    a.year = 2000;
14
15    b.title = strdup("Book b");
16    b.year = 2005;
17 → b.year = 2005;
18
19 → a = b;
20
```



После присваивания

```
C (gcc 4.8, C11)
(known limitations)

1 #include <string.h>
2 #include <stdlib.h>
3
4 int main(void)
5 {
6     struct book_t
7     {
8         char *title;
9         int year;
10    } a = { 0 }, b = { 0 };
11
12    a.title = strdup("Book a");
13    a.year = 2000;
14
15    b.title = strdup("Book b");
16    b.year = 2005;
17
18
19 → a = b;
20
21 → free(a.title);
22    free(b.title);
```



Структуры с полями-указателями

Стратегия так называемого «*глубокого копирования*» (англ., *deep copying*) подразумевает создание копий объектов, на которые ссылаются поля структуры.

```
int book_copy(struct book_t *dst, const struct book_t *src)
{
    char *ptmp = strdup(src->title);
    if (ptmp)
    {
        free(dst->title);
        dst->title = ptmp;
        dst->year  = src->year;

        return 0;
    }

    return 1;
}
```

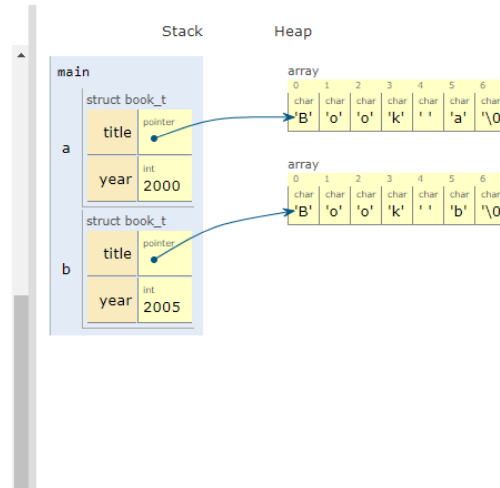
Структуры с полями-указателями

Стратегия «глубокого копирования».

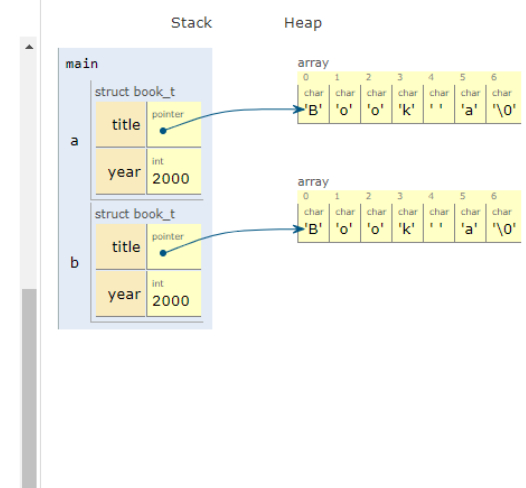
До копирования

После копирования

```
44 C (gcc 4.8, C11)
45 (known limitations)
23 dst->year = src->year;
24
25 return 0;
26 }
27
28 int main(void)
29 {
30     struct book_t a = { 0 }, b = { 0 };
31
32     a.title = strdup("Book a");
33     a.year = 2000;
34
35     b.title = strdup("Book b");
36     b.year = 2005;
37
38     book_copy(&b, &a);
39
40     free(a.title);
41     free(b.title);
42 }
```



```
44 C (gcc 4.8, C11)
45 (known limitations)
23 dst->year = src->year;
24
25 return 0;
26 }
27
28 int main(void)
29 {
30     struct book_t a = { 0 }, b = { 0 };
31
32     a.title = strdup("Book a");
33     a.year = 2000;
34
35     b.title = strdup("Book b");
36     b.year = 2005;
37
38     book_copy(&b, &a);
39
40     free(a.title);
41     free(b.title);
42 }
```



Структуры с полями-указателями

```
struct book_t* book_create(const char *title, int year)
{
    struct book_t *pbook = malloc(sizeof(struct book_t));

    if (pbook)
    {
        pbook->title = strdup(title);
        if (pbook->title)
            pbook->year = year;
        else
        {
            free(pbook);
            pbook = NULL;
        }
    }

    return pbook;
}
```

```
struct book_t *pbook = NULL;

pbook = book_create("Book a", 2000);
if (pbook)
{
    // Работа с книгой

    // Корректно ли так освобождают память?
    free(pbook);
}
```