DEEP & SHALLOW

Поверхностное и глубокое копирование + несколько интересных задач по этим концепциям

К. Владимиров, Intel, 2018
mail-to: konstantin.vladimirov@gmail.com

Что такое копирование?

- В случае простых структур данных, ответ прост: копия побитово равна оригиналу

```
struct S { int x; double y; };
```

- Скопировать такую структуру очень просто

```
struct S s1 = {1, 2.0}, s2;
s2 = s1; // copy
assert(s2.x == 1 && s2.y == 2.0);
```

- Но что если рассмотреть структуру посложнее?

Идея односвязного списка

- Идея односвязного списка довольно проста: каждый узел содержит указатель на следующий

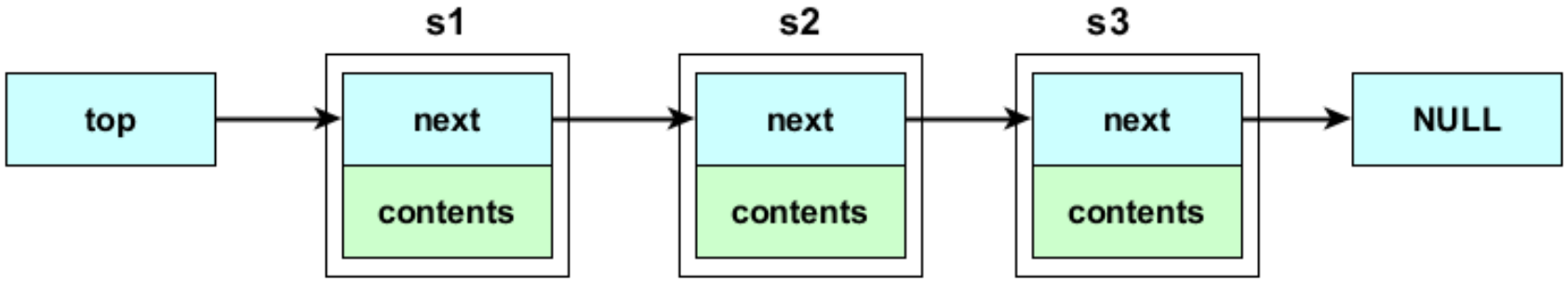
```
struct node_t {struct node_t *next; int contents; };

struct node_t s1, s2, s3, *top;
top = &s1;
s1.next = &s2; s1.contents = 1;
s2.next = &s3; s2.contents = 2;
s3.next = NULL; s3.contents = 3;
```

- Разумеется узлы могут быть выделены где угодно, обычно они в динамической памяти. Здесь они на стеке просто для примера
- Можно изобразить это как рисунок

Идея односвязного списка

- Идея односвязного списка довольно проста: каждый узел содержит указатель на следующий

```
struct node_t {struct node_t *next; int contents; };
```

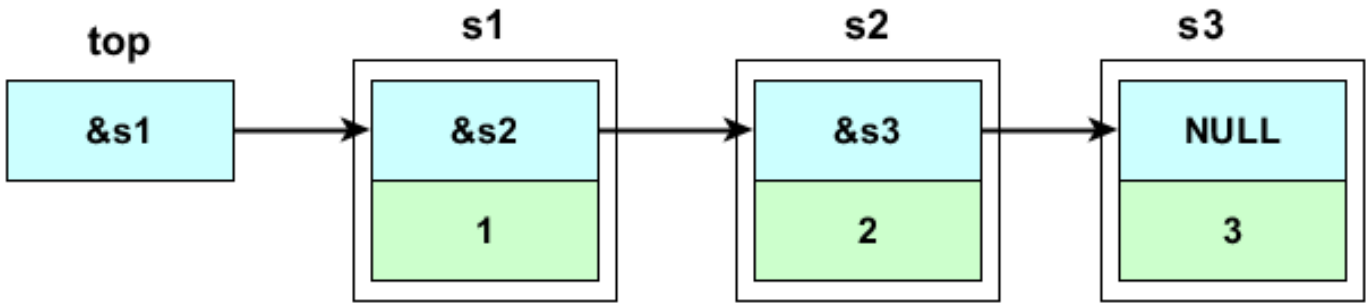


- Картинку можно несколько конкретизировать, учтя значения полей, выставленные на предыдущем слайде

Идея односвязного списка

- Идея односвязного списка довольно проста: каждый узел содержит указатель на следующий

```
struct node_t {struct node_t *next; int contents; };
```



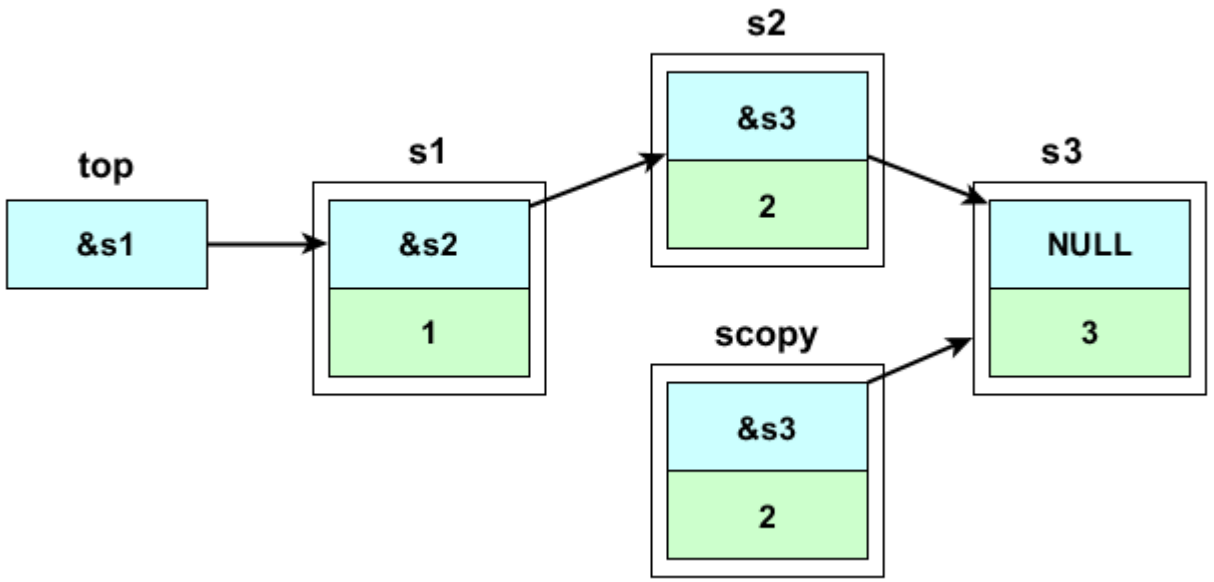
- Что значит скопировать узел s2 односвязного списка?

```
struct node_t scopy = s2; // что здесь происходит?
```

Поверхностное (shallow) копирование

```
struct node_t scopy = s2; // что здесь происходит?
```

- При таком копировании структура копируется побитово. Но биты поля next в s2 это указатель на s3



Поверхностное копирование: проблемы

- Очень часто поверхностное копирование в случае нетривиальных структур это не то, чего бы нам хотелось

```
struct named_t { int num; char *name; };

struct named_t n;
n.num = 1;
n.name = calloc(100); strcpy(n.name, "one");

struct named_t ncopy = n;

free(n.name); // ncopy.name также становится невалидным
```

- Здесь как и в примере стека, поверхностное копирование привело к совместному владению строкой

Глубокое (deep) копирование

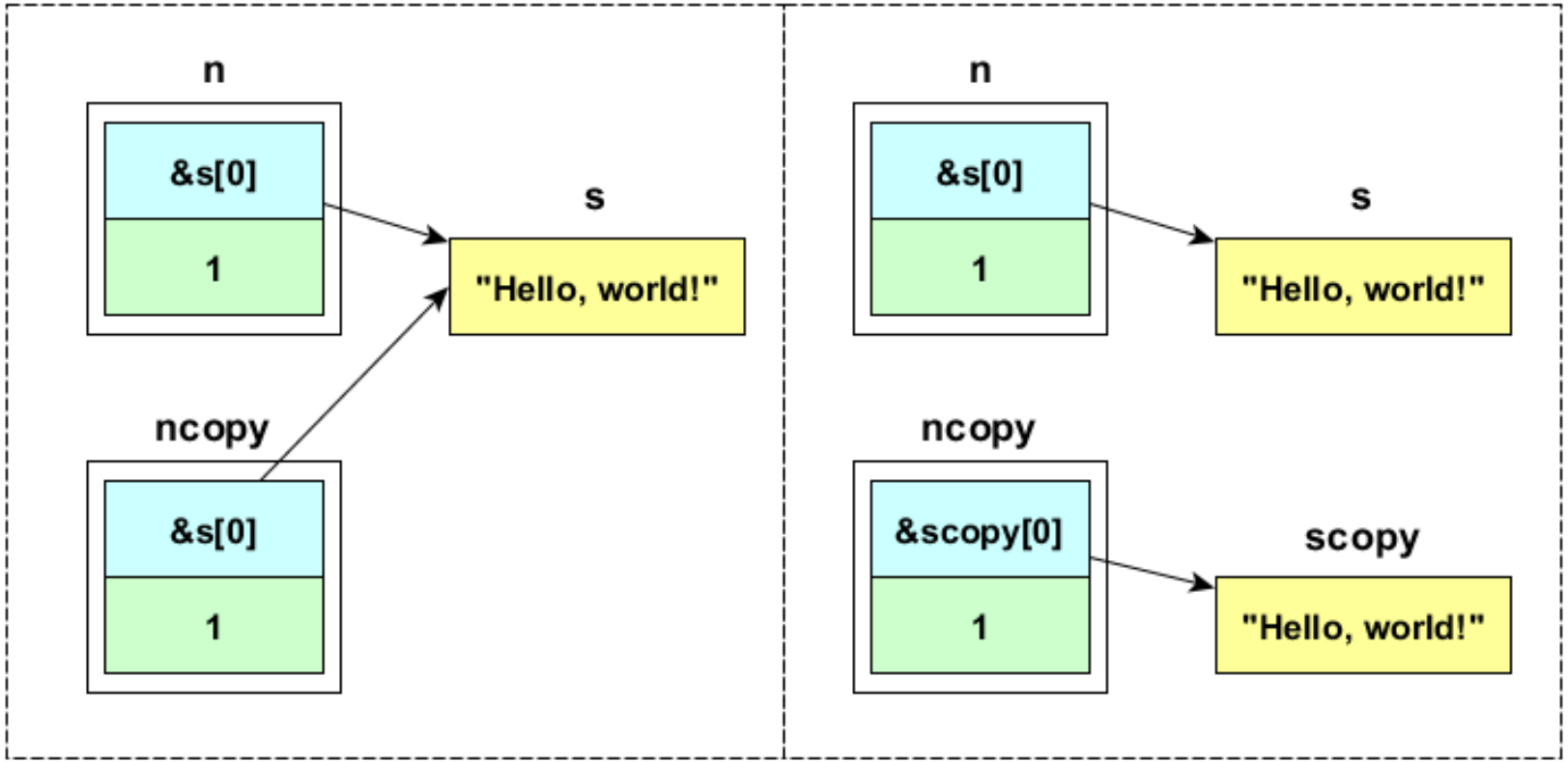
- Глубокое копирование не делается средствами языка, его надо писать

```
struct named_t { int num; char *name; };

struct named_t deepcopy(struct named_t src) {
  struct named_t srccopy;
  srccopy.num = src.num;
  srccopy.name = calloc(strlen(src.name + 1));
  strcpy(srccopy.name, src.name);
  return srccopy;
}
```

- Здесь выделяется новая память и все поля структуры копируются в неё. Разумеется, написать такую функцию можно только зная устройство конкретной структуры

Обсуждение

- Что из двух вариантов ncopy является на самом деле копией?



Обсуждение

- Что из двух вариантов ncopy является на самом деле копией?
- Вопрос философский. Иногда семантика требует глубокого копирования, иногда поверхностного
- Важно знать их отличия и помнить, что поверхностное (побитовое) копирование у нас есть даром и по умолчанию, а глубокое всегда нужно реализовать самостоятельно

Problem DC: глубокое копирование

- Структура namednum объединяет строку и число. Строку можно считать всегда выделенной в динамической памяти

```
struct namednum { char *name; int number; };
```

- Реализуйте функцию, которая копирует массив таких структур, выделяя новую память для строк и копируя туда строки

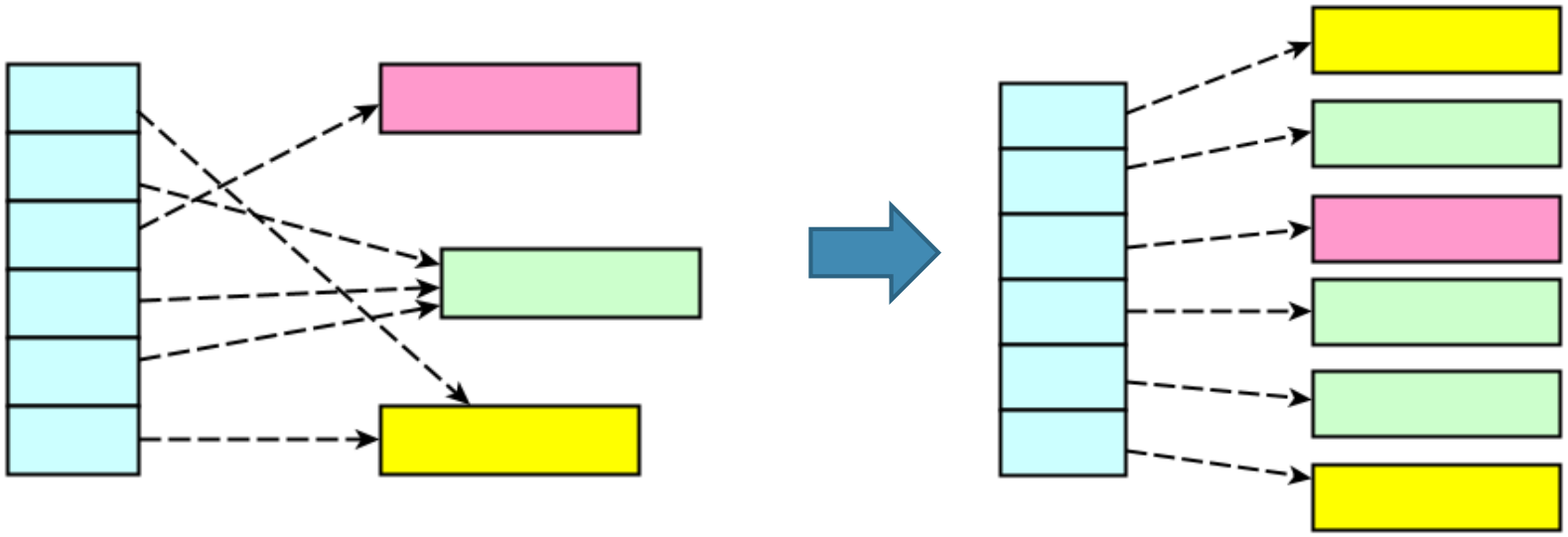
```
void namednumcpy(struct namednum *dst,
                 const struct namednum *src, int srclen) {
  // TODO: ваш код здесь
}
```

- Можно предполагать, что памяти для самого массива dst уже выделено вполне достаточно, но она не инициализирована

Problem AC: исправление указателей

- Структура `namednum` объединяет строку и число. Строку можно считать всегда выделенной в динамической памяти (см. Problem DC). Реализуйте функцию, «исправляющую» указатели, т.е. расшаривающую общие строки в массиве таких структур

```
void fixupstrings(struct namednum *arr, int arrlen);
```



Problem LA: список из массива списков

- Дана структура, являющаяся в том числе узлом односвязного списка

```
struct node { struct node *next; int n; };
```

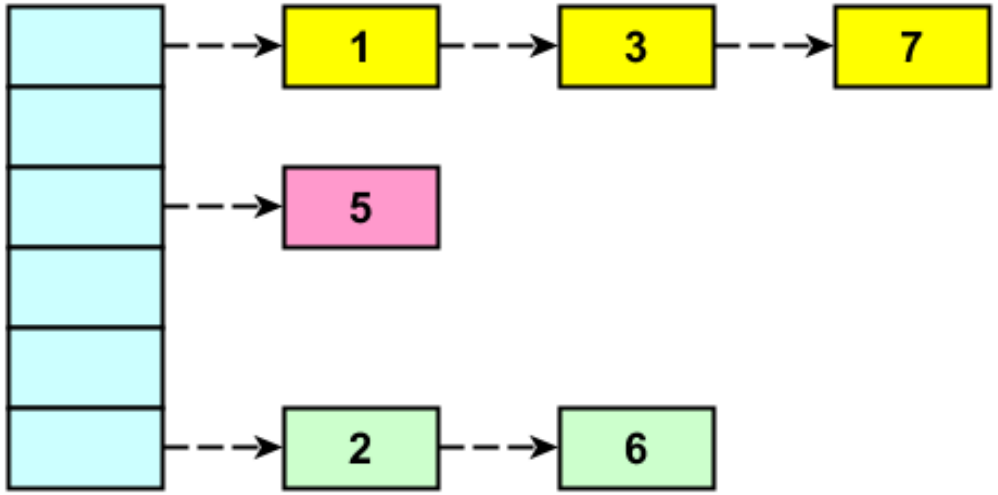
- Реализуйте функцию, которая из массива сортированных списков делает один сортированный список

```
struct node * mergelists(struct node **arr, int arrlen);
```

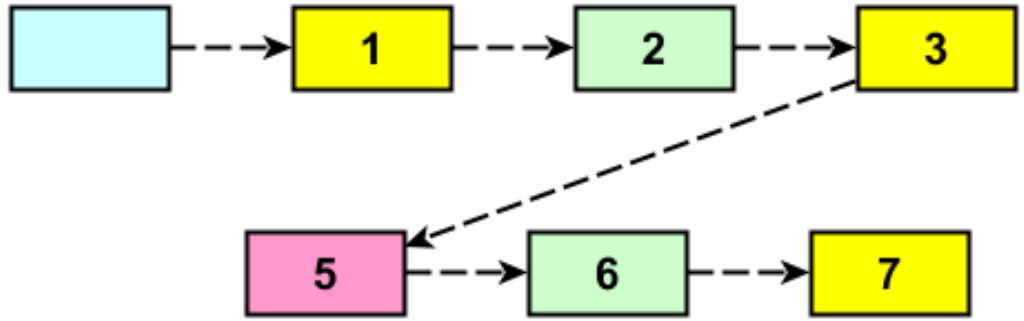
- Обратите внимание: эта проблема вообще не требует копирования и эта функция не должна выделять дополнительной динамической памяти
- На следующем слайде будет поясняющий рисунок

Problem LA: список из массива списков

Состояние «до»



Состояние «после»



- Синим цветом на этих рисунках изображены не узлы, а указатели на узел. Соответственно входной массив arr это массив указателей не узлы и функция mergelists возвращает указатель на первый узел объединённого списка

Problem RV: переворот массива структур

- Реализуйте функцию, которая переворачивает массив структур, не выделяя дополнительной памяти

```
void reverse(void *arr, int size, int n);
```

- Обратите внимание: вы не знаете точное содержимое структуры и должны работать с void pointers. Всё что вам известно это сколько структур и какого они размера
- Дополнительно можно предполагать, что в структуре нет владеющих указателей и таким образом возможно поверхностное копирование

```
struct wrong { int x; int *px; };
```

- Такая структура под условия задачи не подходит (все ли понимают почему?)

Литература

- [C11] ISO/IEC – Information technology – Programming languages – C, 2011
- [$K\&R$] Brian W. Kernighan, Dennis Ritchie – The C programming language, 1988
- [$Linden$] Peter van der Linden – Expert C Programming: Deep C Secrets , 1994
- [$Cormen$] Thomas H. Cormen – Introduction to Algorithms, 2009
- [$TAOCP$] Donald E. Knuth – The Art of Computer Programming, 2011
- [$SALG$] Robert Sedgewick Algorithms, 4th edition, 2011