acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CfaqNullPointer :

Нулевые указатели

- Нулевые указатели
  - 1.1
  - 1.2
  - 1.3
  - 1.4
  - 1.5
  - 1.6
  - 1.7
  - 1.8
  - 1.9
  - 1.10
  - 1.11
  - 1.12
  - 1.13
  - 1.14
  - 1.15

Поиск

Поиск

Раздел «Язык Си»

Главная
Зачем учить С?
Определения

Инструменты:

Поиск
Изменения
Index
Статистика

Разделы

Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C

Logon>>

1.1

Q: Расскажите о нулевых указателях.

A: Для каждого типа указателей существует (согласно определению языка) особое значение - "нулевой указатель", которое отлично от всех других значений и не указывает на какой-либо объект или функцию. Таким образом, ни оператор &, ни успешный вызов `malloc()` никогда не приведут к появлению нулевого указателя. (`malloc` возвращает нулевой указатель, когда память выделить не удается, и это типичный пример использования нулевых указателей как особых величин, имеющих несколько иной смысл "память не выделена" или "теперь ни на что не указываю".)

Нулевой указатель принципиально отличается от неинициализированного указателя. Известно, что нулевой указатель не ссылается ни на какой объект; неинициализированный указатель может ссылаться на что угодно. См. также вопросы 3.1, 3.13, и 17.1.

В приведенном выше определении уже упоминалось, что существует нулевой указатель для каждого типа указателя, и внутренние значения нулевых указателей разных типов могут отличаться. Хотя программистам не обязательно знать внутренние значения, компилятору всегда необходима информация о типе указателя, чтобы различить нулевые указатели, когда это нужно (см. ниже).

Смотри: K&R I Разд. 5.4 с. 97-8; K&R II Разд. 5.4 с. 102; H&S Разд. 5.3 с. 91; ANSI Разд. 3.2.2.3 с. 38.

1.2

Q: Как "получить" нулевой указатель в программе?

A: В языке C константа 0, когда она распознается как указатель, преобразуется компилятором в нулевой указатель. То есть, если во время инициализации, присваивания или сравнения с одной стороны стоит переменная или выражение, имеющее тип указателя, компилятор решает, что константа 0 с другой стороны должна превратиться в нулевой указатель и генерирует нулевой указатель нужного типа. Следовательно, следующий фрагмент абсолютно корректен:

```
char *p = 0;
if(p != 0)
```

Однако, аргумент, передаваемый функции, не обязательно будет распознан как значение указателя, и компилятор может оказаться не способным распознать голый 0

как нулевой указатель. Например, системный вызов UNIX `execl` использует в качестве параметров переменное количество указателей на аргументы, завершаемое нулевым указателем. Чтобы получить нулевой указатель при вызове функции, обычно необходимо явное приведение типов, чтобы 0 воспринимался как нулевой указатель.

```
execl("/bin/sh", "sh", "-c", "ls", (char *)0);
```

Если не делать преобразования `(char *)`, компилятор не поймет, что необходимо передать нулевой указатель и вместо этого передаст число 0. (Заметьте, что многие руководства по UNIX неправильно объясняют этот пример.)

Когда прототипы функций находятся в области видимости, передача аргументов идет в соответствии с прототипом и большинство приведений типов может быть опущено, так как прототип указывает компилятору, что необходим указатель определенного типа, давая возможность правильно преобразовать нули в указатели. Прототипы функций не могут, однако, обеспечить правильное преобразование типов в случае, когда функция имеет список аргументов переменной длины, так что для таких аргументов необходимы явные преобразования типов. Всегда безопаснее явные преобразования в нулевой указатель, чтобы не наткнуться на функцию с переменным числом аргументов или на функцию без прототипа, чтобы временно использовать не-ANSI компиляторы, чтобы продемонстрировать, что Вы знаете, что делаете. (Кстати, самое простое правило для запоминания.)

Итог:

- **Можно использовать 0**
  - инициализация
  - присваивание
  - сравнение
  - вызов функции, прототип в области видимости, количество аргументов фиксировано
- **Необходимо преобразование**
  - вызов функции, прототип которой вне области видимости
  - переменное число аргументов при вызове функции

Смотри: K&R I Разд. A7.7 с. 190, Разд. A7.14 с. 192; K&R II Разд. A7.10 с. 207, Разд. A7.17 с. 209; H&S Разд. 4.6.3 с. 72; ANSI Разд. 3.2.2.3 .

1.3

Q: Что такое NULL и как он определен с помощью `#define`?

A: Многим программистам не нравятся нули, беспорядочно разбросанные по программам. По этой причине макрос препроцессора NULL определен в `<stdio.h>` или `<stddef.h>` как значение 0 (или `(void *) 0`, об этом значении поговорим позже.) Программист, который хочет явно различать 0 как целое и 0 как нулевой указатель может использовать NULL в тех местах, где необходим нулевой указатель. Это только стилистическое соглашение; препроцессор преобразует NULL опять в 0, который затем распознается компилятором в соответствующем контексте как нулевой указатель. В отдельных случаях при передаче параметров функции, может все же потребоваться явное указание типа перед NULL (как и перед 0). (Таблица в вопросе 1.2 приложима как к NULL, так и к 0).

NULL нужно использовать *только* для указателей; см. вопрос 1.8.

Смотри: K&R I Разд. 5.4 с. 97-8; K&R II Разд. 5.4 с. 102; H&S Разд. 13.1 с. 283; ANSI Разд. 4.1.5 с. 99, Разд. 3.2.2.3 с. 38, Rationale Разд. 4.1.5 с. 74.

1.4

Q: Как `#define` должен определять NULL на машинах, использующих ненулевой двоичный код для внутреннего представления нулевого указателя?

A: Программистам нет необходимости знать внутреннее представление(я) нулевых указателей, ведь об этом обычно заботится компилятор. Если машина использует ненулевой код для представления нулевых указателей, на совести компилятора генерировать этот код, когда программист обозначает нулевой указатель как 0 или NULL. Следовательно, определение NULL как 0 на машине, для которой нулевые указатели представляются ненулевыми значениями так же правомерно как и на любой другой, так как компилятор должен (и может) генерировать корректные значения нулевых указателей в ответ на 0, встретившийся в соответствующем контексте

1.5

Q: Пусть NULL был определен следующим образом:

```
#define   NULL ((char *)0)
```

Ознает ли это, что функциям можно передавать NULL без преобразования типа

A: В общем, нет. Проблема в том, что существуют компьютеры, которые используют различные внутренние представления для указателей на различные типы данных. Предложенное определение через `#define` годится, когда функция ожидает в качестве передаваемого параметра указатель на `char`, но могут возникнуть проблемы при передаче указателей на переменные других типов, а верная конструкция

```
FILE *fp = NULL;
```

может не сработать.

Тем не менее, ANSI C допускает другое определение для `NULL`:

```
#define NULL ((void *)0)
```

Кроме помощи в работе некорректным программам (но только в случае машин, где указатели на разные типы имеют одинаковые размеры, так что помощь здесь сомнительна) это определение может выявить программы, которые неверно используют `NULL` (например, когда был необходим символ ASCII NUL; см. вопрос 1.8).

Смотри: ANSI Rationale Разд. 4.1.5 с. 74.

1.6

Q: Я использую макрос

```
#define   Nullptr(type) (type *)0
```

который помогает задавать тип нулевого указателя.

A: Хотя этот трюк и популярен в определенных кругах, он стоит немного. Он не нужен при сравнении и присваивании; см. вопрос 1.2. Он даже не экономит буквы. Его использование показывает тому, кто читает программу, что автор здорово "сечет" в нулевых указателях, и требует гораздо более аккуратной проверки определения макроса, его использования и *всех* остальных случаев применения указателей. См. также вопрос 8.1.

1.7

Q: Корректно ли использовать сокращенный условный оператор `if(p)` для проверки того, что указатель ненулевой? А что если внутреннее представление для нулевых указателей отлично от нуля?

A: Когда C требует логическое значение выражения (в инструкциях `if`, `while`, `for`, и `do` и для операторов `&&`, `||`, `!`, и `?:`) значение `false` получается, когда выражение равно нулю, а значение `true` получается в противоположном случае. Таким образом, если написано

```
   if(expr)
```

где `expr` - произвольное выражение, компилятор на самом деле поступает так, как будто было написано

```
   if(expr != 0)
```

Подставляя тривиальное выражение, содержащее указатель `p` вместо `expr`, получим

`if(p)` эквивалентно `if(p != 0)`

и это случай, когда происходит сравнение, так что компилятор поймет, что неявный ноль - это нулевой указатель и будет использовать правильное значение. Здесь нет никакого подвоха, компиляторы работают именно так и генерируют в обоих случаях идентичный код. Внутреннее представление указателя *не* имеет значения.

Оператор логического отрицания `!` может быть описан так:

`!expr` на самом деле эквивалентно `expr?0:1`

Читателю предлагается в качестве упражнения показать, что

`if(!p)` эквивалентно `if(p == 0)`

Хотя "сокращения" типа `if(p)` совершенно корректны, кое-кто считает их использование дурным стилем.

См. также вопрос 8.2.

Смотри: K&R II Разд. A7.4.7 с. 204; H&S Разд. 5.3 с. 91; ANSI Разд. 3.3.3.3, 3.3.9, 3.3.13, 3.3.14, 3.3.15, 3.6.4.1, и 3.6.5 .

1.8

Q: Если NULL и 0 эквивалентны, то какую форму из двух использовать?

A: Многие программисты верят, что NULL должен использоваться во всех выражениях, содержащих указатели как напоминание о том, что значение должно рассматриваться как указатель. Другие же чувствуют, что путаница, окружающая NULL и 0, только усугубляется, если 0 спрятать в операторе `#define` и предпочитают использовать 0 вместо NULL. Единственного ответа не существует. Программисты на C должны понимать, что NULL и 0 взаимозаменяемы и что 0 без преобразования типа можно без сомнения использовать при инициализации, присваивании и сравнении. Любое использование NULL (в противоположность 0) должно рассматриваться как ненавязчивое напоминание, что используется указатель; программистам не нужно ничего делать (как для своего собственного понимания, так и для компилятора) для того, чтобы отличать нулевые указатели от целого числа 0. NULL *нельзя* использовать, когда необходим другой тип нуля. Даже если это и будет работать, с точки зрения стиля программирования это плохо.(ANSI позволяет определить NULL с помощью `#define` как `(void *)0`. Такое определение не позволит использовать NULL там, где не подразумеваются указатели). Особенно не рекомендуется использовать NULL там, где требуется нулевой код ASCII (NUL). Если необходимо, напишите собственное определение

```
#define NUL '\0'
```

Смотри: K&R II Разд. 5.4 с. 102.

1.9

Q: Но не лучше ли будет использовать NULL (вместо 0) в случае, когда значение NULL изменяется, быть может, на компьютере с ненулевым внутренним представлением нулевых указателей?

A: Нет. Хотя символические константы часто используются вместо чисел из-за того, что числа могут измениться, в данном случае причина, по которой используется NULL, иная. Еще раз повторим: язык гарантирует, что 0, встреченный там, где по контексту подразумевается указатель, будет заменен компилятором на нулевой указатель. NULL используется только с точки зрения лучшего стиля программирования.

1.10

Q: Я в растерянности. Гарантируется, что NULL равен 0, а нулевой указатель нет?

A: Термин "null" (англ. "нуль", "нулевой") или NULL может не совсем обдуманно использоваться в нескольких смыслах:

1. Нулевой указатель как абстрактное понятие языка, определенное в вопросе 1.1. Он представляется с помощью...
2. Внутреннее (на стадии выполнения) представление нулевого указателя, которое может быть отлично от нуля и различаться для различных типов указателей. О внутреннем представлении нулевого указателя должны заботиться только создатели компилятора. Программистам на C это представление не известно, поскольку они используют...
3. Синтаксическое соглашение для нулевых указателей, символ **0**. Вместо него часто используют...
4. Макрос NULL который с помощью #define определен как **0** или "(void *)=0=". Наконец, нас может запутать...
5. Нулевой код ASCII (NUL), в котором все биты равны нулю, но который имеет мало общего с нулевым указателем, разве что названия похожи; и...

6. "Строка нулевой длины" ("нулевая строка"), или, что то же самое, пустая строка (""). Термин "нулевая строка" может приводить к путанице в C и, возможно, его следует избегать, так как пустая строка включает символ '\0', но не нулевой указатель, и здесь мы уже идем по кругу...

В этом документе фраза "нулевой указатель" (прописными буквами) используется в смысле 1, символ **0** в смысле 3, а слово **NULL**, записанное большими буквами, в смысле 4.

1.11

Q: Почему так много путаницы связано с нулевыми указателями? Почему так часто возникают вопросы?

A: Программисты на C традиционно хотят знать больше, чем это необходимо для программирования, о внутреннем представлении кода. Тот факт, что внутреннее представление нулевых указателей для большинства машин совпадает с их представлением в исходном тексте, т.е. нулем, способствует появлению неверных обобщений. Использование макроса (NULL) предполагает, что значение может впоследствии измениться, или иметь другое значение для какого-нибудь компьютера. Конструкция `if(p == 0)` может быть истолкована неверно, как преобразование перед сравнением p к целому типу, а не 0 к типу указателя. Наконец, часто не замечают, что термин "null" (в англоязычной литературе) употребляется в разных смыслах (перечисленных выше).

Хороший способ устранить путаницу - вообразить, что язык C имеет ключевое слово (возможно, nil, как в Паскале), которое обозначает нулевой указатель. Компилятор либо пребразует "nil" в нулевой указатель нужного типа, либо сообщает об ошибке, когда этого сделать нельзя. На самом деле, ключевое слово для нулевого указателя в C - это не "nil" а "0". Это ключевое слово работает всегда, за исключением случая, когда компилятор воспринимает в неподходящем контексте "0" без указания типа как целое число, равное нулю, вместо того, чтобы сообщить об ошибке. Программа может не работать, если предполагалось, что "0" без явного указания типа - это нулевой указатель.

1.12

Q: Я все еще в замешательстве. Мне так и не понятна возня с нулевыми указателями.

A: Следуйте двум простым правилам:

1. Для обозначения в исходном тексте нулевого указателя, используйте 0 или NULL.
2. Если 0 или NULL используются как фактические аргументы при вызове функции, приведите их к типу указателя, который ожидает вызываемая функция.

Остальная часть дискуссии посвящена другим заблуждениям, связанным с нулевыми указателями, внутреннему представлению нулевых указателей (которое Вам знать не обязательно), а также усовершенствованиям стандарта ANSI C. Изучите ответы на вопросы 1.1, 1.2, и 1.3, а также учтите вопросы 1.8 и 1.11, и все будет нормально.

1.13

Q: Учитывая всю эту путаницу, связанную с нулевыми указателями, не лучше ли просто потребовать, чтобы их внутреннее представление было нулевым?

A: Если причина только в этом, то поступать так было бы неразумно, так как это неоправданно ограничит конкретную реализацию, которая (без таких ограничений) будет естественным образом представлять нулевые указатели специальными, отличными от нуля значениями, особенно когда эти значения автоматически будут вызывать специальные аппаратные прерывания, связанные с неверным доступом.

Кроме того, что это требование даст на практике? Понимание нулевых указателей не требует знаний о том, нулевое или ненулевое их внутреннее представление. Предположение о том, что внутреннее представление нулевое, не приводит к упрощению кода (за исключением некоторых случаем сомнительного использования `calloc`; см. вопрос 3.13). Знание того, что внутреннее представление равно нулю, не упростит вызовы функций, так как *размер* указателя может быть отличным от размера указателя на int. (Если вместо 0 для обозначения нулевого указателя

использовать "nil" (см. вопрос 1.11), необходимость в нулевом внутреннем представлении нулевых указателей даже бы не возникла).

1.14

Q: Ну а если честно, на какой-нибудь реальной машине используются ненулевые внутренние представления нулевых указателей или разные представления для указателей разных типов?

Серия Prime 50 использует сегмент 07777, смещение 0 для нулевого указателя, по крайней мере, для PL/I. Более поздние модели используют сегмент 0, смещение 0 для нулевых указателей C, что делает необходимыми новые инструкции, такие как TCNP (проверить нулевой указатель C), которые вводятся для совместимости с уцелевшими скверно написанными C программами, основанными на неверных предположениях. Старые машины Prime с адресацией слов были печально знамениты тем, что указатели на байты (char *) у них были большего размера, чем указатели на слова (int *).

Серия Eclipse MV корпорации Data General имеет три аппаратно поддерживаемых типа указателей (указатели на слово, байт и бит), два из которых - char * и void * используются компиляторами C. Указатель word * используется во всех других случаях.

Некоторые центральные процессоры Honeywell-Bull используют код 06000 для внутреннего представления нулевых указателей.

Серия CDC Cyber 180 использует 48-битные указатели, состоящие из кольца (ring), сегмента и смещения. Большинство пользователей (в кольце 11) имеют в качестве нулевых указателей код 0xB00000000000.

Символическая Лисп-машина с теговой архитектурой даже не имеет общеупотребительных указателей; она использует пару (вообще говоря, несуществующий <объект, смещение> хендл) как нулевой указатель C.

В зависимости от модели памяти, процессоры 80*86 (PC) могут использовать либо 16-битные указатели на данные и 32-битные указатели на функции, либо, наоборот, 32-битные указатели на данные и 16-битные - на функции.

Старые модели HP 3000 используют различные схемы адресации для байтов и для слов. Указатели на char и на void, имеют, следовательно, другое представление, чем указатели на int (на структуры и т.п.), даже если адрес одинаков.

1.15

Q: Что означает ошибка во время исполнения "null pointer assignment" (запись по нулевому адресу). Как мне ее отследить?

Это сообщение появляется только в системе MS-DOS (см., следовательно, раздел 16) и означает, что произошла запись либо с помощью неинициализированного, либо нулевого указателя в нулевую область.

Отладчик обычно позволяет установить точку останова при доступе к нулевой области. Если это сделать нельзя, Вы можете скопировать около 20 байт из области 0 в другую и периодически проверять, не изменились ли эти данные.

Примечание редактора: В операционных системах Windows невозможна запись по нулевому адресу. В этом случае программа прекращает свою работу. В UNIX/Linux принят более параноидальный подход: если писать по нулевому адресу нельзя, то и чтение по нулевому адресу - это предвестник записи, т.е. истинный корень проблемы, а запись по нулевому адресу - ее результат. Поэтому в этих системах *чтение* по нулевому адресу так же приводит к принудительному завершению программы. Отладчик вам в руки.

-- TatyanaDerbysheva - 03 Jan 2011