acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CfaqPointer :

Указатели и массивы

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>>

2.1

Q: В одном файле у меня есть описание `char a[6]` а в другом я объявил `extern char *a`. Почему это не работает?

A: Декларация `extern char *a` просто не совпадает с текущим определением. Тип *Указатель-на-тип-T* не равен типу *массив-типа-T*. Используйте `extern char a[]`.

2.2

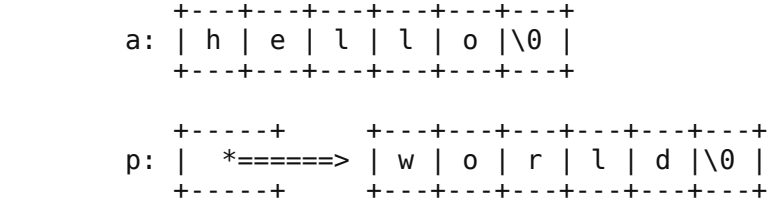
Q: Но я слышал, что `char a[]` эквивалентно `char *a`.

A: Ничего подобного. (То, что Вы слышали, касается формальных параметров функций, см. вопрос 2.4.) Массивы - не указатели. Объявление массива `char a[6];` требует определенного места для шести символов, которое будет известно под именем `a`. То есть, существует место под именем `a`, в которое могут быть помещены 6 символов. С другой стороны, объявление указателя `char *p;` требует места только для самого указателя. Указатель будет известен под именем `p` и может указывать на любой символ (или непрерывный массив символов).

Как обычно, лучше один раз увидеть, чем сто раз услышать. Объявление

```
char a[] = "hello";
char *p = "world";
```

породит структуры данных, которые могут быть представлены так:



Важно понимать, что ссылка типа `x[3]` порождает разный код в зависимости от того, массив `x` или указатель. Если взять приведенную выше декларацию, то, когда компилятор встречается с выражением `a[3]`, он генерирует код, позволяющий

переместиться к месту под именем a, перемещается на три символа вперед и затем читает требуемый символ. В случае выражения p[3] компилятор генерирует код, чтобы начать с позиции p, считывает значение указателя, прибавляет к указателю 3 и, наконец, читает символ, на который указывает указатель. В нашем примере и a[3] и p[3] оказались равны l, но компилятор получает этот символ по-разному. (Смотри также вопросы 17.19 и 17.20.)

2.3

Q: Тогда что же понимается под "эквивалентностью указателей и массивов" в C?

A: Большая часть путаницы вокруг указателей в C происходит от непонимания этого утверждения. "Эквивалентность" указателей и массивов не позволяет говорить не только об идентичности, но и взаимозаменяемости.

"Эквивалентность" относится к следующему ключевому определению:

значение [см. вопрос 2.5] типа массив-T, которое появляется в выражении, превращается (за исключением трех случаев) в указатель на первый элемент массива; тип результирующего указателя - указатель-на-T .

(Исключение составляют случаи, когда массив оказывается операндом sizeof, оператора & или инициализатором символьной строки для массива литер.)

Вследствие этого определения нет заметной разницы в поведении оператора индексирования [], если его применять к массивам и указателям. Согласно правилу, приведенному выше, в выражении типа a[i] ссылка на массив a превращается в указатель и дальнейшая индексация происходит так, как будто существует выражение с указателем p[i] (хотя доступ к памяти будет различным, как описано в ответе на вопрос 2.2). В любом случае выражение x[i], где x - массив или указатель) равно по определению *((x)+(i)).

Смотри: K&R I Разд.5.3 с.93-6; K&R II Разд.5.3 с. 99; H&S Разд.5.4.1 с. 93; ANSI Разд.3.2.2.1, Разд.3.3.2.1, Разд.3.3.6 .

2.4

Q: Тогда почему объявления указателей и массивов взаимозаменяемы в в качестве формальных параметров?

A: Так как массивы немедленно превращаются в указатели, массив на самом деле не передается в функцию. По общему правилу, любое похожее на массив объявление параметра

```
f(a)
char a[];
```

рассматривается компилятором как указатель, так что если был передан массив, функция получит:

```
f(a)
char *a;
```

Это превращение происходит только для формальных параметров функций, больше нигде. Если это превращение раздражает Вас, избегайте его; многие пришли к выводу, что порождаемая этим путаница перевешивает небольшое преимущество от того, что объявления смотрятся как вызов функции и/или напоминают о том, как параметр будет использоваться внутри функции.

Смотри: K&R I Разд.5.3 с. 95, Разд.A10.1 с. 205; K&R II Разд.5.3 с. 100, Разд.A8.6.3 с. 218, Разд.A10.1 с.226; H&S Разд.5.4.3 с. 96; ANSI Разд.3.5.4.3, Разд.3.7.1, CT&P Разд.3.3 с. 33-4.

2.5

Q: Как массив может быть значением типа lvalue, если нельзя присвоить ему значение?

A: Стандарт ANSI C определяет "модифицируемое lvalue", но массив к этому не относится.

Смотри: ANSI Разд. 3.2.2.1 с. 37.

2.6

Q: Почему sizeof неправильно определяет размер массива, который передан функции в качестве параметра?

A: Оператор sizeof сообщает размер указателя, который на самом деле получает функция. (см. вопрос 2.4).

2.7

Q: Кто-то объяснил мне, что массивы это на самом деле только постоянные (неизменяемые) указатели.

A: Это слишком большое упрощение. Имя массива – это константа, следовательно, ему нельзя присвоить значение, но массив – это не указатель, как должно быть ясно из ответа на вопрос 2.2 и из картинки, помещенной там же.

2.8

Q: С практической точки зрения в чем разница между массивами и указателями?

A: Массивы автоматически резервируют память, но не могут изменить расположение в памяти и размер. Указатель должен быть задан так, чтобы явно указывать на выбранный участок памяти (возможно с помощью malloc), но он может быть по нашему желанию переопределен (т.е. будет указывать на другие объекты) и, кроме того, указатель имеет много других применений, кроме службы в качестве базового адреса блоков памяти.

В рамках так называемой эквивалентности массивов и указателей (см. вопрос 2.3), массивы и указатели часто оказываются взаимозаменяемыми. Особенно это касается блока памяти, выделенного функцией malloc, указатель на который часто используется как настоящий массив. (На этот блок памяти можно ссылаться, используя [], см. вопрос 2.14, а также вопрос 17.20.)

2.9

Q: Я наткнулась на шуточный код, содержащий "выражение" 5["abcdef"]. Почему такие выражения возможны в C?

A: Да, Вирджиния, индекс и имя массива можно переставлять в C. Этот забавный факт следует из определения индексации через указатель, а именно, a[e] идентично *((a)+(e)), для *любого* выражения e и основного выражения a, до тех пор пока одно из них будет указателем, а другое целочисленным выражением. Это неожиданная коммутативность часто со странной гордостью упоминается в C-текстах, но за пределами Соревнований по Непонятному Программированию (Obfuscated C Contest) она применения не находит. (см. вопрос 17.13).

Смотри: ANSI Rationale Разд. 3.3.2.1 с. 41.

Примечание редактора: Подобный трюк используется при программировании крутящегося прогрессора в текстовом режиме.

```
    for(i=0; ; i=(i+1)%4 )
         printf("\b%c", "|/-\"[i]);
```

2.10

Q: Мой компилятор ругается, когда я передаю двумерный массив функции, ожидающей указатель на указатель.

A: Правило, по которому массивы превращаются в указатели не может применяться рекурсивно. Массив массивов (т.е. двумерный массив в C) превращается в указатель на массив, а не в указатель на указатель. Указатели на массивы могут вводить в заблуждение и применять их нужно с осторожностью. (Путаница еще более усугубляется тем, что существуют некорректные компиляторы, включая некоторые версии pcc и полученные на основе pcc программы lint, которые неверно воспринимают присваивание многоуровневым указателям многомерных массивов.) Если вы передаете двумерный массив функции:

```
   int array[NROWS][NCOLUMNS];
   f(array);
```

описание функции должно соответствовать

```
    f(int a[][NCOLUMNS]) {...}
```
или
```
    f(int (*ap)[NCOLUMNS]) {...}   /* ap - указатель на массив */
```

В случае, когда используется первое описание, компилятор неявно осуществляет обычное преобразование "массива массивов" в "указатель на массив"; во втором случае указатель на массив задается явно. Так как вызываемая функция не выделяет место для массива, нет необходимости знать его размер, так что количество "строк" NROWS может быть опущено. "Форма" массива по-прежнему важна, так что размер "столбца" NCOLUMNS должен быть включен (а для массивов размерности 3 и больше, все промежуточные размеры).

Если формальный параметр функции описан как указатель на указатель, то передача функции в качестве параметра двумерного массива будет, видимо, некорректной.

Смотри: K&R I Разд.5.10 с. 110; K&R II Разд.5.9 с. 113.

2.11

Q: Как писать функции, принимающие в качестве параметра двумерные массивы, "ширина" которых во время компиляции неизвестна?

A: Это непросто. Один из путей - передать указатель на элемент [0][0] вместе с размерами и затем симулировать индексацию "вручную":

```
f2(int * aryp, int nrows, int ncolumns)
{ ... array[i][j] это aryp[i * ncolumns + j] ... }
```

Этой функции массив из вопроса 2.10 может быть передан так:

```
f2(&array[0][0], NROWS, NCOLUMNS);
```

Нужно, однако, заметить, что программа, выполняющая индексирование многомерного массива "вручную" не полностью соответствует стандарту ANSI C; поведение (&array[0][0])[x] не определено при x > NCOLUMNS.

gcc разрешает объявлять локальные массивы, которые имеют размеры, задаваемые аргументами функции, но это - нестандартное расширение. См. также вопрос 2.15.

2.12

Q: Как объявить указатель на массив?

A: Обычно этого делать не нужно. Когда случайно говорят об указателе на массив, обычно имеют в виду указатель на первый элемент массива.

Вместо указателя на массив рассмотрим использование указателя на один из элементов массива. Массивы типа T превращаются в указатели типа T (см. вопрос 2.3), что удобно; индексация или увеличение указателя позволяет иметь доступ к отдельным элементам массива. Истинные указатели на массивы при увеличении или индексации указывают на следующий массив и в общем случае если и полезны, то лишь при операциях с массивами массивов. (См. вопрос 2.10 выше.)

Если действительно нужно объявить указатель на целый массив, используйте что-то вроде int (*ap)[N]; где N - размер массива. (См. также вопрос 10.4.) Если размер массива неизвестен, параметр N может быть опущен, но получившийся в результате тип "указатель на массив неизвестного размера" - бесполезен.

2.13

Q: Исходя из того, что ссылки на массив превращаются в указатели, скажите в чем разница между array и &array для массива

```
    int array[NROWS][NCOLUMNS];
```

A: Согласно ANSI/ISO стандарту C, &array дает указатель типа "указатель-на-массив-T", на весь массив (См. также вопрос 2.12). В языке C до выхода стандарта ANSI оператор & в &array игнорировался, порождая предупреждение компилятора. Все компиляторы C, встречая просто имя массива, порождают указатель типа указатель-на-T, т.е. на первый элемент массива. (См. также вопрос 2.3.)

2.14

Q: Как динамически выделить память для многомерного массива?

A: Лучше всего выделить память для массива указателей, а затем инициализировать каждый указатель так, чтобы он указывал на динамически создаваемую строку. Вот пример для двумерного массива:

```
int **array1 = (int **)malloc(nrows * sizeof(int *));
for(i = 0; i < nrows; i++)
   array1[i] = (int *)malloc(ncolumns * sizeof(int));
```

(В "реальной" программе, `malloc` должна быть правильно объявлена, а каждое возвращаемое `malloc` значение - проверено.)

Можно поддерживать монолитность массива, (одновременно затрудняя последующий перенос в другое место памяти отдельных строк), с помощью явно заданных арифметических действий с указателями:

```
int **array2 = (int **)malloc(nrows * sizeof(int *));
array2[0] = (int *)malloc(nrows * ncolumns * sizeof(int));
for(i   = 1; i < nrows;   i++)
   array2[i] = array2[0] + i * ncolumns;
```

В любом случае доступ к элементам динамически задаваемого массива может быть произведен с помощью обычной индексации: `array[i][j]`.

Если двойная косвенная адресация, присутствующая в приведенных выше примерах, Вас по каким-то причинам не устраивает, можно имитировать двумерный массив с помощью динамически задаваемого одномерного массива:

```
int *array3 = (int *)malloc(nrows * ncolumns * sizeof(int));
```

Теперь, однако, операции индексирования нужно выполнять вручную, осуществляя доступ к элементу `i,j` с помощью `array3[i*ncolumns+j]`. (Реальные вычисления можно скрыть в макросе, однако вызов макроса требует круглых скобок и запятых, которые не выглядят в точности так, как индексы многомерного массива.)

Наконец, можно использовать указатели на массивы:

```
int (*array4)[NCOLUMNS] =
      (int(*)[NCOLUMNS])malloc(nrows * sizeof(*array4));,
```

но синтакс становится устрашающим, и "всего лишь" одно измерение должно быть известно во время компиляции.

Пользуясь описанными приемами, необходимо освобождать память, занимаемую массивами (это может проходить в несколько шагов; см. вопрос 3.9), когда они больше не нужны, и не следует смешивать динамически создаваемые массивы с обычными, статическими (см. вопрос 2.15 ниже, а также вопрос 2.10).

2.15

Q: Как мне равноправно использовать статически и динамически задаваемые многомерные массивы при передаче их в качестве параметров функциям?

A: Идеального решения не существует. Возьмем объявления

```
int array[NROWS][NCOLUMNS];
int **array1;
int **array2;
int *array3;
int (*array4)[NCOLUMNS];
```

соответствующие способам выделения памяти в вопросах 2.10 и 2.14, и функции, объявленные как

```
f1(int a[][NCOLUMNS], int m, int n);
f2(int *aryp, int nrows, int ncolumns);
f3(int **pp, int m, int n);
```

(см. вопросы 2.10 и 2.11). Тогда следующие вызовы должны работать так, как ожидается

```
f1(array, NROWS, NCOLUMNS);
f1(array4, nrows, NCOLUMNS);
f2(&array[0][0], NROWS, NCOLUMNS);
f2(*array2, nrows, ncolumns);
f2(array3, nrows, ncolumns);
f2(*array4, nrows, NCOLUMNS);
f3(array1, nrows, ncolumns);
f3(array2, nrows, ncolumns);
```

Следующие два вызова, возможно, будут работать, но они включают сомнительные приведения типов, и работают лишь в том случае, когда динамически задаваемое число столбцов ncolumns совпадает с NCOLUMS:

```
f1((int (*)[NCOLUMNS])(*array2), nrows, ncolumns);
f1((int (*)[NCOLUMNS])array3, nrows, ncolumns);
```

Необходимо еще раз отметить, что передача &array[0][0] функции f2 не совсем соответствует стандарту; см. вопрос 2.11.

Если Вы способны понять, почему все вышеперечисленные вызовы работают и написаны именно так, а не иначе, и если Вы понимаете, почему сочетания, не попавшие в список, работать не будут, то у Вас *очень* хорошее понимание массивов и указателей (и нескольких других областей) C.

2.16

Q: Вот изящный трюк: если я пишу

```
int realarray[10];
int *array = &realarray[-1];,
```

*то теперь можно рассматривать array как массив, у которого индекс первого элемента равен единице. *

A: Хотя этот прием внешне привлекателен (и использовался в старых изданиях книги "Numerical Recipes in C"), он не удовлетворяет стандартам C. Арифметические действия над указателями определены лишь тогда, когда указатель ссылается на выделенный блок памяти или на воображаемый завершающий элемент, следующий сразу за блоком. В противном случае поведение программы не определено, *даже если указатель не переназначается*. Код, приведенный выше, плох тем, что при уменьшении смещения может быть получен неверный адрес (возможно, из-за циклического перехода адреса при пересечении границы сегмента).

Смотри: ANSI Разд.3.3.6 с. 48, Rationale Разд.3.2.2.3 с. 38; K&R II Разд.5.3 с. 100, Разд.5.4 с. 102-3, Разд.A7.7 с. 205-6.

2.17

Q: Я передаю функции указатель, который она инициализирует

```
...
int *ip;
f(ip);
...

void f(int * ip)
{
   static int dummy = 5;
   ip = &dummy;
}
```

но указатель после вызова функции остается неизменным.

A: Функция пытается изменить сам указатель, или то, на что он ссылается? Помните, что аргументы в C передаются по значению. Вызываемая функция изменяет только копию передаваемого указателя. Вам нужно либо передать адрес указателя (функцию будет в этом случае принимать указатель на указатель), либо сделать так, чтобы функция возвращала указатель.

2.18

Q: У меня определен указатель на char, который указывает еще и на int, причем мне необходимо переходить к следующему элементу типа int. Почему не работает

```
((int *)p)++;
```

A: В языке C оператор преобразования типа не означает "будем действовать так, как будто эти биты имеют другой тип"; это оператор, который действительно выполняет преобразования, причем по определению получается значение типа *rvalue*, которому нельзя присвоить новое значение и к которому не применим оператор ++. (Следует считать аномалией то, что компиляторы pcc и расширения gcc вообще воспринимают выражения приведенного выше типа.). Скажите то, что думаете:

```
p = (char *)((int *)p + 1);
```

или просто

```
p += sizeof(int);
```

Смотри: ANSI Разд.3.3.4, Rationale Разд.3.3.2.4 с. 43.

2.19

Q: Могу я использовать void ** , чтобы передать функции по ссылке обобщенный указатель?

A: Стандартного решения не существует, поскольку в C нет общего типа указатель-на-указатель. `void *` выступает в роли обобщенного указателя только потому, что автоматически осуществляются преобразования в ту и другую сторону, когда встречаются разные типы указателей. Эти преобразования не могут быть выполнены (истинный тип указателя неизвестен), если осуществляется попытка косвенной адресации, когда `void **` указывает на что-то отличное от `void *`.

-- TatyanaDerbysheva - 04 Jan 2011