acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CfaqMemory :

Выделение памяти

3.1

Q: Почему не работает фрагмент кода?

```
char *answer;
printf("Type something:\n");
gets(answer);
printf("You typed \"%s\"\n", answer);
```

A: Указатель answer, который передается функции gets как место, в котором должны храниться вводимые символы, не инициализирован, т.е. не указывает на какое-то выделенное место. Иными словами, нельзя сказать, на что указывает answer. (Так как локальные переменные не инициализируются, они вначале обычно содержат "мусор", то есть даже не гарантируется, что в начале answer - это нулевой указатель. См. вопрос 17.1).

Простейший способ исправить программу - использовать локальный массив вместо указателя, предоставив компилятору заботу о выделении памяти:

```
#include

char answer[100], *p;
printf("Type something:\n");
fgets(answer, sizeof(answer), stdin);
if((p = strchr(answer, '\n')) != NULL)
   *p = '\0';
printf("You typed \"%s\"\n", answer);
```

Заметьте, что в этом примере используется fgets() вместо gets() (это признак хорошего тона, см. вопрос 11.6), что позволяет указать размер массива, так что выход за пределы массива, когда пользователь введет слишком длинную строку, становится невозможным. (К сожалению, fgets() не удаляет автоматически завершающий символ конца строки \n, как это делает gets()). Для выделения памяти можно также использовать malloc().

3.2

Q: Не могу заставить работать strcat. Код приводит к странным результатам:

```
char *s1 = "Hello, ";
char *s2 = "world!";
char *s3 = strcat(s1, s2);
```

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>>

A: Проблема снова состоит в том, что не выделено место для результата объединения. С не поддерживает автоматически переменные типа "строка". Компиляторы С выделяют память только под объекты, явно указанные в исходном тексте (в случае строк это может быть массив литер или символы, заключенные в двойные кавычки). Программист должен сам позаботиться о том, чтобы была выделена память для результата, который получается в процессе выполнения программы, например результата объединения строк. Обычно это достигается объявлением массива или вызовом `malloc`. (см. также вопрос 17.20).

Функция `strcat` не выделяет память; вторая строка присоединяется к первой. Следовательно, одно из исправлений - в задании первой строки в виде массива достаточной длины

```
char s1[20] = "Hello, ";
```

Так как strcat возвращает указатель на первую строку (в нашем случае `s1`), переменная `s3` - лишняя.

Смотри: CT&P Разд. 3.2 с. 32.

3.3

Q: Но в справочнике о функции `strcat` сказано, что она использует в качестве аргументов два указателя на `char`. Откуда мне знать о выделении памяти?

A: Как правило, при использовании указателей *всегда* необходимо иметь в виду выделение памяти, по крайней мере, быть уверенным, что компилятор делает это для Вас. Если в документации на библиотечную функцию явно ничего не сказано о выделении памяти, то обычно это проблема вызывающей функции.

Краткое описание функции в верхней части страницы справочника в стиле UNIX может ввести в заблуждение. Приведенные там фрагменты кода ближе к определению, необходимому для разработчика функции, чем для того, кто будет эту функцию вызывать. В частности, многие функции, имеющие в качестве параметров указатели (на структуры или стринги, например), обычно вызываются с параметрами, равными адресам каких-то уже существующих объектов(структур или массивов - см. вопросы 2.3, 2.4.) Другой распространенный пример - функция `stat()`.

3.4

Q: Предполагается, что функция, которую я использую, возвращает строку, но после возврата в вызывающую функцию эта строка содержит "мусор".

A: Убедитесь, что правильно выделена область памяти, указатель на которую возвращает Ваша функция. Функция должна возвращать указатель на статически выделеннную область памяти или на буфер, передаваемый функции в качестве параметра, или на память, выделенную с помощью `malloc()`, но *не* на локальный (auto) массив. Другими словами, никогда никогда не делайте ничего похожего на

```
char *f()
{
   char   buf[10];
   /* ... */
   return buf;
}
```

Приведем одну поправку (непригодную в случае, когда `f()` вызывается рекурсивно, или когда одновременно нужны несколько возвращаемых значений)

```
      static char buf[10];
```

См. также вопрос 17.5.

3.5

Q: Почему в некоторых исходных текстах значения, возвращаемые `malloc()`, аккуратно преобразуются в указатели на выделяемый тип памяти?

A: До того как стандарт ANSI/ISO ввел обобщенный тип указателя `void *`, эти преобразования были обычно необходимы для подавления предупреждений компилятора о приравнивании указателей разных типов. (Согласно стандарту C ANSI/ISO, такие преобразования типов указателей не требуются).

3.6

Q: Можно использовать содержимое динамически выделяемой памяти после того как она освобождена?

A: Нет. Иногда в старых описаниях malloc() говорилось, что содержимое освобожденной памяти "остается неизменным"; такого рода поспешная гарантия никогда не была универсальной и не требуется стандартом ANSI.

Немногие программисты стали бы нарочно использовать содержимое освобожденной памяти, но это легко сделать нечаянно. Рассмотрите следующий (корректный) фрагмент программы, в котором освобождается память, занятая односвязным списком:

```
struct list *listp, *nextp;
for(listp = base; listp != NULL; listp = nextp) {
   nextp = listp->next;
   free((char *)listp);
}
```

и подумайте, что получится, если будет использовано на первый взгляд более очевидное выражение для тела цикла listp = listp->next, без временного указателя nextp.

См.: ANSI Rationale Разд. 4.10.3.2 с. 102; CT&P Разд. 7.10 с. 95.

3.7

Q: Откуда free() знает, сколько байт освобождать?

A: Функции malloc/free запоминают размер каждого выделяемого и возвращаемого блока, так что не нужно напоминать размер освобождаемого блока.

3.8

Q: А могу я узнать действительный размер выделяемого блока?

A: Нет универсального ответа.

3.9

Q: Я выделяю память для структур, которые содержат указатели на другие динамически создаваемые объекты. Когда я освобождаю память, занятую структурой, должен ли я сначала осводить память, занятую подчиненным объектом?

A: Да. В общем, необходимо сделать так, чтобы каждый указатель, возвращаемый malloc() был передан free() точно один раз (если память освобождается).

3.10

Q: В моей программе сначала с помощью malloc() выделяется память, а затем большое количество памяти освобождается с помощью free(), но количество занятой памяти (так сообщает команда операционной системы) не уменьшается.

A: Большинство реализаций malloc/free не возвращают освобожденную память операционной системе (если таковая имеется), а просто делают освобожденную память доступной для будущих вызовов malloc() в рамках того же процесса.

3.11

Q: Должен ли я освобождать выделенную память перед возвратом в операцинную систему?

A: Делать это не обязательно. Настоящая операционная система восстанавливает состояние памяти по окончании работы программы. Тем не менее, о некоторых персональных компьютерах известно, что они ненадежны при восстановлении памяти, а из стандарта ANSI/ISO можно лишь получить указание, что эти вопросы относятся к "качеству реализации".

См. ANSI Разд. 4.10.3.2 .

3.12

Q: Правильно ли использовать нулевой указатель в качестве первого аргумента функции realloc()? Зачем это нужно?

A: Это разрешено стандартом ANSI C (можно также использовать realloc(...,0) для освобождения памяти), но некоторые ранние реализации C это не поддерживают, и мобильность в этом случае не гарантируется. Передача нулевого указателя realloc() может упростить написание самостартующего алгоритма пошагового выделения памяти.

См. ANSI Разд. 4.10.3.4 .

3.13

Q: В чем разница между calloc и malloc? Получатся ли в результате применения calloc корректные значения нулевых указателей и чисел с плавающей точкой? Освобождает ли free память, выделенную calloc, или нужно использовать cfree?

A: По существу calloc(m,n) эквивалентна

```
p = malloc(m * n);
memset(p, 0, m * n);
```

Заполнение нулями означает зануление всех битов, и, следовательно, не гарантирует нулевых значений для указателей (см. раздел 1) и для чисел с плавающей точкой. Функция free может (и должна) использоваться для освобождения памяти, выделенной calloc.

Смотри: ANSI Разделы от 4.10.3 до 4.10.3.2 .

3.14

Q: Что такое alloca и почему использование этой функции обескураживает?

A: alloca выделяет память, которая автоматически освобождается, когда происходит возврат из функции, в которой вызывалась alloca. То есть, память, выделенная alloca, локальна по отношению к "стековому кадру" или контексту данной функции.

Использование alloca не может быть мобильным, реализации этой функции трудны на машинах без стека. Использование этой функции проблематично (и очевидная реализация на машинах со стеком не удается), когда возвращаемое ей значение непосредственно передается другой функции, как, например, в fgets(alloca(100), 100, stdin).

По изложенным выше причинам alloca (вне зависимости от того, насколько это может быть полезно) нельзя использовать в программах, которые должны быть в высокой степени мобильны.

См. ANSI Rationale Разд. 4.10.3 с. 102.

-- TatyanaDerbysheva - 06 Jan 2011