Lesson 14

|  |  |
| --- | --- |
| Коллеги, здравствуйте.  Рад всех приветствовать на последнем уроке курса Основ Языка С.  Это занятие последнее, но не по важности. Сильная сторона языка С не только в возможности работать с указателями, но и в возможности самостоятельно управлять выделяемой памятью внутри программы. В языках высокого уровня данная возможность зачастую скрыта от программиста, чтобы по случайности не подвесить среду виртуализации и не сломать операционную систему. |  |
| Итак, как мы уже знаем, все переменные всех типов как-то хранятся в памяти, и до этого момента нас устраивало, как операционная система нам эту память выделяет. Но, пришло время взять бразды правления в свои руки и начать самостоятельно брать то, что наше по праву. Шучу, конечно я о памяти. Процесс выделения памяти для программы называется memory allocation отсюда и название функции, которая выделяет память и пишет в предложенный идентификатор указатель на начало этой области. malloc(size); - *она принимает в качестве аргумента размер выделяемой памяти.* | int main (int argc, const char\* argv[]){  //void\* malloc(<memory size>);  return 0;  } |
| Как видим, функция возвращает пустоту, то есть область памяти будет зафиксирована, но не размечена. То есть это будет некоторая пустая область из n байт. Чтобы иметь возможность в этой области хранить значения нам нужно её подготовить для хранения этих значений – разметить.  Например, мы уверены, что будем складывать в нашу область памяти какие-то целые числа типа integer. Для этого при вызове функции нам надо использовать синтаксис приведения типа полученной области. *Мы знаем, что каждая переменная типа integer хрпнится в памяти в 4 байтах. Например мы создаем указатель на некоторую область состоящую из 123 байт – таким образом будет выделена память для 123 байт, но она никак не будет размечена.*  *А при помощи оператора приведения типа мы скажем компилятору*, что нам необходимо выделить некоторую область памяти, поделить её на ячейки размера int, и каждой такой ячейке дать свой адрес. | int variable;  int \*area = (int\*) malloc(123); |
| *Итак, как узнать сколько нужно выделить памяти и при этом не запутаться. Для этого не обязательно знать размеры всех типов данных языка Си, для этого придумали оператор sizeof. Оператор sizeоf возвращает размер переменной (типа данных) в байтах. Мы напишем sizeof (int), умножим его на 10. Таким образом мы выделим память в размере 40 байт и разметим их под хранение переменных типа int. Фактически мы сделали то же самое что и описание массива типа int при помощи записи объявления массива с использованием квадратных скобок.* | int \*area = (int\*) malloc(sizeof (int) \* 10);  int array[10]; |
| Помните, мы говорили про арифметику указателей? Вот это то место, где она нам поможет понять, что вообще происходит. *Давайте реализуем два массива привычным нам способом и при помощи динамического выделения памяти. Для реализации массива нам понадобится его размер, определим его как константу SIZE. Заменим в объявленном массиве 10 на SIZE и заполним этот массив какими-нибудь значениями.*  *И напишем второй цикл для вывода этого массива в консоль.*  *Добавим пустую строку*  *И проделаем то же самое со вторым массивом, который мы инициализировали как область памяти. В этом виде данный код демонстрирует что мы можем одинаков работать и с массивами, объявленными привычными способами и с динамически выделенной областью памяти. Для более наглядной разницы, при заполнении и выводе в консоль второго массива, воспользуемся арифметикой указателей.*  *Запустим наш проект, все работает.* | const int SIZE = 10;  int \*area = (int\*) malloc(sizeof (int) \* SIZE);  int array [SIZE];  for(i = 0; i < SIZE; i++)  array [i] = i \* 10;  for(i = 0; i < SIZE; i++)  printf("%d ", array [i]);  puts("");  for(i = 0; i < SIZE; i++) area[i] = i\*10;  for(i = 0; i < SIZE; i++) printf("%d ", area[i]);  for(i = 0; i < SIZE; i++) \*(area + i) = i \* 10;  for(i = 0; i < SIZE; i++) printf("%d ", \*(area + i)); |
| Напомню, мы реализовали массив area вручную. то есть выполняем ровно те операции, которые выполняет компилятор при реализации массива array. Разложили синтаксис на базовые операции. Очевидно, что это знание открывает нам возможность распределять память в нашем приложении для любых типов данных, в том числе и сложных, таких как структуры. Для того, чтобы каждый раз не пересчитывать размеры переменных вручную, особенно это актуально для строк и структур, используют оператор sizeof(), который возвращает целочисленное значение в байтах, которое займёт в памяти та или иная переменная. |  |
| Функция malloc резервирует память для нашей программы, но делает это весьма просто, вместе со всеми теми случайными переменными, которые могут там храниться.  *Давайте я продемонстрирую это. Закомментируем заполнение массива area, и увидим в консоли непонятные ненулевые значения.*  Для того чтобы гарантированно очистить вновь выделенную область памяти используют функцию calloc() clear allocate которая не только выделит нам память, но и очистит содержимое. Поскольку функция не только выделяет память но и очищает её, считается, что она работает медленнее, чем malloc. Синтаксис её весьма похож, только размеры необходимой области памяти передаются двумя аргументами - первый - сколько элементов, второй - какого размера будут элементы. | int \*area = (int\*) calloc(SIZE, sizeof (int)); |
| По окончании работы с областью памяти надо её освободить, чтобы ОС могла её использовать по своему усмотрению. Делается это при помощи функции free(). Если не освобождать память после использования - велика вероятность того, что мы, например, в каком-то цикле будем выделять себе место под какую-то структуру, и рано или поздно съедим всю память. Неприятно может получиться. | free(area); |
| И напоследок: довольно часто возникают ситуации, когда нам нужно придать нашей программе какой-то динамики, в этом случае мы можем изменить размеры уже выделенного блока памяти. Например, расширить наш массив, добавив туда пару элементов. Это делается при помощи функции realloc() в которую мы должны передать указатель на область памяти, которую хотим изменить, и размеры новой области памяти. При помощи этой функции области памяти можно как увеличивать, так и уменьшать. | area = realloc(area, sizeof (int)); |
| Большинство компиляторов выделяют память блоками, размеры которых обычно равны какой-то из степеней двойки, поэтому при объявлении или изменении области памяти в 20 байт, скорее всего будет выделена область в 32 байта, или если мы объявим 70 байт, то скорее всего будет выделено 128. То есть при работе с областями памяти не стоит ожидать, что они будут даваться нашей программе подряд. Организация памяти это отдельный долгий разговор, явно выходящий за рамки Основ.  *Давайте запустим нашу программу с вновь выделенным измененным блоком памяти и увидим что все прекрасно перевыделилось.* | puts("");  int newsize = SIZE + 10;  for(i = 0; i < newsize; i++) \*(area + i) = i \* 10;  for(i = 0; i < newsize; i++) printf("%d ", \*(area + i)); |
| Спасибо, за внимание и интерес, проявленный к курсу. За прошедшие 14 уроков мы узнали как устроена практически любая программа изнутри, научились работать с памятью и указателями, узнали основные принципы и механизмы работы программ на уровне операционной системы. Заглянули внутрь привычных синтаксических конструкций, узнали, что делают и что скрывают от программистов среды виртуализации и фреймворки. Я и команда Гикбрейнс желает всем успехов в освоении Ваших профессий.  Удачи и до новых встреч, а ну да, и не забывайте освобождать память! | free(area); // don’t forget to free allocated memory))) |