Указатели

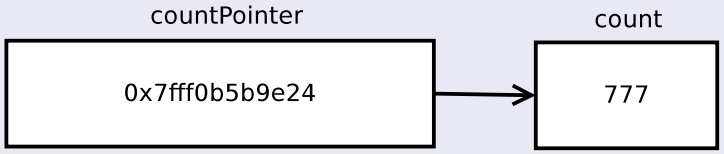
Лектор: Артамонов Юрий Николаевич

Университет "Дубна"
филиал Котельники

Содержание

Понятие указателя

Указатели являются наиболее мощным элементом языка C. Указатель - это переменная, значением которой является адрес памяти. В обычной переменной в свою очередь содержится ее значение. Указатель же содержит адрес этой переменной, в которой хранится ее значение. Говорят, что переменная непосредственно ссылается на значение, указатель косвенно ссылается на значение. Указатель, также как любая переменная, должен быть объявлен до первого использования. Чтобы объявить переменную-указатель, перед его именем ставится звездочка *.



Рассмотрим пример:

Пример объявленния указателя

```
  #include <stdio.h>
main()
{
  int *countPointer, count;
  scanf("%d",&count);
  countPointer = &count;
  printf("Значение переменной count равно %d непосредственная( ссылка)\n",
  count);
  printf("Значение указателя на переменную count равно %p адрес(
  переменной count в памяти)\n", countPointer);
  printf("Значение физического адреса переменной count равно %p\n", &
  count);
  printf("Значение переменной count с использованием косвенной адресации
  равно %d\n",*countPointer);
  return 0;
}
```

Объяснение примера

Как видно из примера, символ * в объявлении распространяется только на countPointer. Этот символ означает, что объявленная переменная является указателем (в данном случае указатель countPointer ссылается на объект целого типа). Можно объявлять указатели, ссылающиеся на объекты любого типа. Указатели должны быть инициализированы либо при объявлении, либо при помощи оператора присваивания. Указатель может быть инициализирован нулем, макросом NULL или значением адреса. Указатель со значением NULL ни на что не указывает (символическая константа NULL определяется в заголовочном файле stdio.h). Присваивание значения 0 эквивалентно присваиванию NULL. Чтобы присвоить указателю адрес какой-либо переменной, используется символ &.

```
countPointer = &count;
```

Наконец, чтобы получить значение переменной, хранящееся по адресу, который является значением указателя, необходимо использовать символ *. Такая операция называется разыменование указателя. Операции * и & взаимно дополняют друг друга

```
printf("Значение переменной count с использованием косвенной адресации равно
  %d\n",*countPointer);//разыменование указателя
```

Операции * и & взаимно дополняют друг друга, если они обе применяются к указателю, то их порядок не имеет значения, в обоих случаях - это все равно, что просто значение указателя countPointer

```
printf("%p адрес( переменной count в памяти)\n",&*countPointer);
```

Передача параметра по ссылке

Как уже отмечалось, существует два способа передачи параметров функции - *передача параметров по значению*, *передача параметров по ссылке*. Без специальных мер все вызовы функций используют передачу параметров по значению (за исключением передачи массивов). Это означает, что в функцию передается не сама переменная, а ее копия. Поэтому любые изменения переменной внутри функции не отражается на изменении значения самой переменной. Это удобно, однако, если в функцию передаются большие структуры данных (например, массивы), то передача по значению связана с копированием большого объема памяти, что приводит к накладным расходам как по памяти, так и по быстродействию программы. Использование указателей позволяет принудительно реализовать передачу параметров функции по ссылке. Рассмотрим пример.

Пример передачи параметра по ссылке

```
#include <stdio.h>
void sqr (int *); //объявление в прототипе типа аргумента – указателя
main()
{
  int n;
  printf("Введите число: ");
  scanf("%d",&n);
  sqr(&n);
  printf("Квадрарт числа равен %d\n", n);
  return 0;
}
void sqr(int *nP)
{
  *nP =*nP * *nP;
}
//Введите число: 7
//Квадрарт числа равен 49
```

Передача параметра по ссылке (продолжение)

Если функция может получить в качестве аргумента одномерный массив, то в функциональном заголовке и в прототипе функции сооответствующий параметр может быть определен как указатель. Компилятор не делает различия между функцией, имеющей параметром указатель, и функцией, которая получает в качестве аргумента одномерный массив. Однако, функция должна знать, когда она получает массив, а когда ссылку на одиночную переменную. Когда компилятор встречает в качестве параметра одномерный масив в форме `int a[]`, он преобразует этот параметр к виду `int *a`. Эти две формы являются взаимозаменяемыми.

Использование модификатора const с указателями

Модификатор const дает возможность сообщить компилятору о том, что значение указанной переменной не должно изменяться.
Существует четыре варианта использования const с указателями:

- изменяемый указатель на изменяемые данные;
- неизменяемый указатель на изменяемые данные;
- изменяемый указатель на неизменяемые данные;
- неизменяемый указатель на неизменяемые данные.

Каждая из этих четырех комбинаций обеспечивает различный уровень привилегий доступа к данным.
Самый высокий уровень доступа: изменяемый указатель на изменяемые данные. Рассмотрим в качестве примера преобразования строки в верхний регистр. Строка будет обрабатываться посимвольно, если символ находится в диапазоне от «a» до «z», то он преобразуется в верхний регистр. Заметим, что ASCII код всех символов верхнего регистра на 32 меньше соответствующих ASCII кодов нижнего регистра.

Пример преобразования строки к верхнему регистру

```
#include<stdio.h>
void convert_to_up(char *);
#define size 10
main(){
   char str[size];int i=0;
   printf("Введите строку:");
   while ((i<size-1)&&((str[i]=getchar()) != EOF)) i++;
   str[size-1] = '\0';
   printf("Исходная строка: %s\n",str); convert_to_up(str);
   printf("Преобразованная строка: %s\n", str);
   return 0;
}
void convert_to_up(char *s){
   while (*s != '\0'){
       if (*s >= 'a' && *s <= 'z') *s -= 32;
       s++;
     }
}
```

Пример работы с изменяемым указателем и неизменяемыми данными

В этом режиме значение указателя может меняться, он может ссылаться на другие данные того же типа, но сами данные на которые он может ссылаться изменяться не могут. Такой режим следует использовать, если нужно пройти по всем элементам некоторого массива, но элементы массива менять нельзя. Классический пример: вывод строки с использованием указателей. В этом случае в параметрах функции указатель объявляется так:

```
... const type *data
```

Рассмотрим подходящий пример.

Пример печати строки без ее случайного изменения

```
#include<stdio.h>
void print_str(const char *);
main()
{
  char str[] = "This program is printing string...";
  print_str(str);
  putchar('\n'); /*Вывод на экран символа */

  return 0;
}
void print_str(const char *s)
{
  for ( ; *s != '\0'; s++) putchar(*s); /*Цикл не инициализируется */
}
```

Работа с неизменяемым указателем и изменяемыми данными

Указатель константа на не-константные данные - это указатель, который всегда указывает на одно и тоже место памяти, а расположенные там данные могут меняться. Например, имя массива является указателем-константой на начало массива. Ко всем элементам массива можно обращаться и изменять их, однако нельзя изменить адрес, который зарезервирован за переменной имени массива. Константные указатели очевидно должны быть инициализированы сразу при объявлении. Константные указатели объявляются как:

```
... type * const ptr
```

Такое объявление должно читаться справа налево и в данном случае оно будет означать, что ptr - это неизменяемый указатель на переменную типа type.
Рассмотрим пример попытки изменить константный указатель.

Пример попытки изменить указатель - константу

```
#include<stdio.h>
main()
{
   int x,y;
   int * const ptr  = &x;
   ptr = &y;
   return 0;
}

//error: assignment of read-only variable ``ptr
```

Работа с неизменяемым указателем и неизменяемыми данными

Минимальные права доступа предоставляет указатель - константа на константные данные. Такой указатель всегда указывает на одно и тоже место в памяти, а расположенные по этому адресу данные не могут модифицироваться. Примером такой ситуации является передача массива функции, которая может только просматривать его элементы, но не может изменять его элементы. Объявление с такими правами доступа выглядит так:

```
... const type * const ptr
```

Рассмотрим пример попытки модифицировать данные или указатель в этой ситуации.

Пример попытки изменить указатель - константу

```c
#include<stdio.h>
main()
{
  int x=10,y;
  const int * const ptr = &y;
  *ptr = x;
  ptr = &x;
  return 0;
}
//error: assignment of read-only location '*'ptr
//error: assignment of read-only variable ''ptr
```

В C имеется специальная унарная операция sizeof, при помощи которой можно определить размер массива или любого другого типа данных в байтах во время компиляции программы. Следующий пример демонстрирует работу с этой функцией.

Работа с функцией sizeof

```
{#include<stdio.h>
main()
{
  printf("Количество байт целого типа int: %d\n",sizeof(int));
  printf("Количество байт короткого целого типа short: %d\n",sizeof(short));
  printf("Количество байт беззнакового целого типа unsigned int: %d\n",sizeof(
  unsigned int));
  printf("Количество байт длинного целого типа long int: %d\n",sizeof(long int
  ));
  printf("Количество байт беззнакового длинного целого типа unsigned long int: %d
  \n",sizeof(unsigned long int));
  printf("Количество байт вещественного типа float: %d\n",sizeof(float));
  printf("Количество байт вещественного типа повышенной точности double: %d\n",
  sizeof(double));
  printf("Количество байт длинного вещественного типа повышенной точности long
  double: %d\n",sizeof(long double));
  printf("Количество байт символьного типа char: %d\n",sizeof(char));
  return 0;
}
```

Работа с функцией sizeof

С использованием sizeof, зная тип массива, можно определить количество его элементов. Это деменстрирует следующая программа:

```
#include<stdio.h>
main()
{
  float a[5];
  printf("Количество элементов массива %d\n", sizeof(a)/sizeof(float));
  return 0;
}
```

Задание 7.1

Реализовать пузырьковую сортировку с использованием указателей

Задание 7.2

Заяц и черепаха

Заяц и черепаха начинают гонку по дороге из 70 квадратов. Старт в 1 квадрате. Победителем является тот, кто первый проскочит финишную линию в 70 квадратов. Победитель вознаграждается ведром свежей моркови и салата. Трасса проходит по склону и иногда соперники проскальзывают, скатываясь вниз. Движение животных регулируется случайным образом в соответствии с таблицей:

Животное	Тип движения	Процент	Описание
Черепаха	тащится быстро	50%	3 квадрата вправо
Черепаха	сползание	20%	6 квадратов влево
Черепаха	тащится медленно	30%	1 квадрат вправо
Заяц	спячка	20%	движения нет
Заяц	большой прыжок	20%	9 квадратов вправо
Заяц	большое сползание	10%	12 квадратов влево
Заяц	малый прыжок	30%	1 квадрат вправо
Заяц	малое сползание	20%	2 квадрата влево

Виды операций над указателями

Над указателями можно выполнять арифметические операции, операции присваивания и сравнения. Однако здесь есть свои особенности.
К указателям может быть применен ограниченный набор арифметических операций: указаель может быть увеличен (++) или уменьшен (- -), к указателю может быть прибавлено целое число (+ или +=), из указателя можно вычесть целое число (- или -=), а также можно вычислить разность двух указателей (обратите внимение, складывать указатели нельзя, просто эта операция не имеет смысла). Рассмотрим пример.

Пример арифметических операций с указателями

```
#include <stdio.h>
main()
{
   int *Ptr1, *Ptr2, a[4] ={10,20,30,40} ;
   Ptr1 = &a[0]; Ptr2 = &a[1];
   printf("Первый указатель: %p, Второй указатель: %p\n", Ptr1, Ptr2);
   printf("Значение первого указателя: %d, Значение второго указателя: %d\n"
   , *Ptr1, *Ptr2);
   Ptr1++; Ptr2--;
   printf("Первый указатель: %p, Второй указатель: %p\n", Ptr1, Ptr2);
   printf("Значение первого указателя: %d, Значение второго указателя: %d\n"
   , *Ptr1, *Ptr2);
}
/* Первый указатель: 0x7ffdfe22a720, Второй указатель: 0x7ffdfe22a724 */
/* Значение первого указателя: 10, Значение второго указателя: 20 */
/* Первый указатель: 0x7ffdfe22a724, Второй указатель: 0x7ffdfe22a720 */
/* Значение первого указателя: 20, Значение второго указателя: 10 */
```

Разбор примера арифметических операций с указателями

Как видно из приведенного примера, увеличение указателя на единицу, или его уменьшение на единицу компилятор понимает не буквально, как с числами. При прибавлении или вычитании из указателя целого числа значение его увеличивается или уменьшается не на это число, а на произведение этого числа на размер объекта на который указатель ссылается.

Так в нашем случае поскольку указатели ссылаются на числа целого типа, который занимает в памяти 4 байта, увеличение указателя на единицу, приводит у его увеличению на эти 4 байта, и наоборот, уменьшение указателя на единицу приводит к его уменьшению на 4 байта. Фактически указанное действие в данном примера эквивалентно смене ссылок указателей.

Пояснение для арифметических операций инкремента, декремента с указателями

Каждый из следующих операторов увеличивает значение указателя, который будет ссылаться на следующий элемент массива:

```
Ptr1++;
++Ptr1;
```

Каждый из следующих операторов уменьшает значение указателя, который будет ссылаться на предыдущий элемент массива:

```
Ptr1--;
--Ptr1;
```

Естественно, изменять значение указателя можно не только на единицу, но и на любое другое число. Рассмотрим соответствующий пример.

Еще один пример арифметических операций с указателями

```
#include <stdio.h>
main()
{
   int *Ptr1, *Ptr2, a[4] ={10,20,30,40} ;
   Ptr1 = &a[0]; Ptr2 = &a[3];
   printf("Первый указатель: %p, Второй указатель: %p\n", Ptr1, Ptr2);
   printf("Значение первого указателя: %d, Значение второго указателя: %d\n"
   , *Ptr1, *Ptr2);
   Ptr1+=2; Ptr2-=3;
   printf("Первый указатель: %p, Второй указатель: %p\n", Ptr1, Ptr2);
   printf("Значение первого указателя: %d, Значение второго указателя: %d\n"
   , *Ptr1, *Ptr2);
}
/* Первый указатель: 0x7ffc01304300, Второй указатель: 0x7ffc0130430c */
/* Значение первого указателя: 10, Значение второго указателя: 40 */
/* Первый указатель: 0x7ffc01304308, Второй указатель: 0x7ffc01304300 */
/* Значение первого указателя: 30, Значение второго указателя: 10 */
```

Вычитание указателей

Еще одной полезной операцией над указателями является операция их вычитания.

```
x = Ptr2-Ptr1;
```

Например, в данном случае переменной x будет присвоено число элементов массива, расположенных начиная с адреса Ptr2 и до адреса Ptr1.
Обычно арифметичекие операции с указателями используют именно при работе с массивами, поскольку их элементы хранятся в памяти последовательно друг за другом.

Присваивание указателей, указатели типа void

Указатель может быть присвоен другому указателю, если оба указателя имеют один и тот же тип. В противном случае нужно использовать операцию приведения типа указателя в правой части оператора присваивания к типу указателя в левой части. Исключением из этого правила является указатель на void (т.е. типа void *), который является обобщенным указателем и может представлять любой тип указателя. Указатель любого типа может быть присвоен указателю на void, и void-указатель может быть присвоен указателю любого типа. В обоих случаях приведение типа не требуется.

Указатель типа void нельзя разыменовать, поскольку заранее неизвестно, сколько байт памяти занимает тот объект, на который ссылается этот указатель.

Сравнение указателей

Указатели могут сравниваться друг с другом при помощи операций сравнения и отношения, но сравнение указателей обычно имеет смысл только если они ссылаются на элементы одного и того же массива. При сравнении указателей сравниваются их адреса, являющиеся значением указателей. Для массива это можно использовать для определения, что один указатель на элемент с большим значением индекса, чем другой. Другая часто используемая операция сравнения - это проверка, не равно ли значение указателя NULL.
Рассмотрим соответствующий пример.

Пример присваивания, сравнения, вычитания указателей

```
#include <stdio.h>
main()
{
   system("clear");
   float *Ptr1, *Ptr2, *Ptr3, a[4] ={1.1,2.2,3.3,4.4};
   void *Ptr4;
   Ptr1 = &a[0]; Ptr2 = &a[3];
   printf("Первый указатель: %p, Второй указатель: %p\n", Ptr1, Ptr2);
   printf("Значение первого указателя: %.1f, Значение второго указателя: %.1
   f\n", *Ptr1, *Ptr2);
   printf ("Число элементов массива между индесами 0 и 3: %d\n",Ptr2-Ptr1
   );
   if (Ptr1<Ptr2) printf("0<3\n"); else printf("Чтото- пошло не так");
   Ptr1 = Ptr2; Ptr4 = Ptr2; Ptr3 = Ptr4;
   printf ("Ptr1 =%p, Ptr2 =%p, Ptr3 =%p\n", Ptr1, Ptr2, Ptr3);
```

Пример присваивания, сравнения, вычитания указателей (продолжение)

```
    Ptr4 = NULL;
    if (Ptr4 == NULL) printf("Ptr4 сброшен в NULL\n");
    printf("Попытка разыменовать Ptr3 %.1f\n",*Ptr3);
}
/* Первый указатель: 0x7ffc2b9887c0, Второй указатель: 0x7ffc2b9887cc */
/* Значение первого указателя: 1.1, Значение второго указателя: 4.4 */
/* Число элементов массива между индесами 0 и 3: 3 */
/* 0<3 */
/* Ptr1 = 0x7ffc2b9887cc, Ptr2 = 0x7ffc2b9887cc, Ptr3 = 0
   x7ffc2b9887cc */
/* Ptr4 сброшен в NULL */
/* Попытка разыменовать Ptr3 4.4 */
```

Связь между массивами и указателями

Как видно из приведенных примеров, массивы и указатели тесно сзаимосвязаны. Имя массива можно рассматривать как указатель-константу. В этом случае можно использовать с указателем индексные выражения. Рассмотрим пример:

```
int a[4]={0}, *Ptr;
Ptr = a; //после этого указатель ссылается на первый элемент массива
printf("%d", a[3]);
//А это альтернативный доступ к элементу массива с индексом 3
 printf("%d", *(Ptr+3));
```

В выражении *(Ptr+3) константа 3 называется смещением. Когда указатель ссылается на начало массива, величина смещения указывает, на какой элемент массива производится ссылка; значение смещения равно значению индекса массива. Приведенный способ записи носит название нотации *указатель/смещение*. В этом выражении использованы круглые скобки, потому что операция * имеет больший приоритет, чем +. Без скобок значение 3 было бы прибавлено к значение *Ptr.

Связь между массивами и указателями (продолжение)

Более того, поскольку имя массива - это указатель на его первый элемент, то его можно использовать по аналогии с указателями. Например, выражение

```
*(a+3)
```

будет ссылаться на элемент массива a[3]. Вообще все выражения и индексами массива могут быть преобразованы в выражения с указателем и смещением. В этом случае в качестве указателя можно использовать имя массива. Указатели в свою очередь, могут использоваться вместо имен массивов в индексных выражениях. Например, выражение:

```
Ptr[1]
```

представляет собой ссылку на элемент массива a[1]. Такой способ можно назвать методом указатель/индекс.
Имя массива - это указатель - константа и он всегда указывает на начало массива. Поэтому выражение

```
a+=1;
```

является недопустимым. Рассмотрим ряд примеров.

Пример индексации массивов с использованием указателей

```
#include <stdio.h>
main()
{
   system("clear");
   int *ptr, a[5] = {1,2,3,4,5},i;
   for (i = 0; i<5; i++)
      printf(" a[%d] =%d", i,*(a+i));
   printf("\n");
   ptr=a;
   for (i = 0; i<5; i++)
      printf(" a[%d] =%d", i,*(ptr+i));
   printf("\n");
   return 0;
}
```

Пример индексации массивов с использованием указателей

```
#include <stdio.h>
char * copy(char *, const char *);
main()
{
  system("clear");
  char str1[20], str2[] = "Hello ",str3[] = "world!", *s;
  s=str1;
  s = copy(s, str2);
  s = copy(s, str3);
  printf("%s\n", str1);
  printf("Длина строки %d\n", s-str1);
}
char * copy(char *s1, const char *s2)
{
  for (; *s1 = *s2; s1++, s2++);
  return s1;
}
```

Указатели на функции

Указатели можно использовать не только, чтобы указывать на какие-либо переменные, элементы массивов в памяти компьютера. Есть возможность ссылаться на функции. *Указатель на функцию* - это переменная, содержащая адрес в памяти, по которому расположена функция. Аналогичным образом имя функции - это адрес начала программного кода функции.

Указатели на функции могут быть переданы функциям в качестве аргументов, могут возвращаться функциями, сохраняться в массивах и присваиваться другим указателям на функции.

Чтобы проиллюстрировать использование указателей на функции, рассмотрим следующий пример: по запросу пользователя требуется выполнить заданную арифметическую операцию (сложение, вычитание, умножение). Используем для этого указатели на функции.

Пример использования указателей на функции

```
#include <stdio.h>
int sum(int, int);
int sub(int, int);
int mul(int,int);
int sum(int a, int b) {return a+b; }
int sub(int a, int b) {return a-b; }
int mul(int a, int b) {return a*b; }
main()
{
  int (*a[3])(int,int)={sum, sub, mul};
  int x,y,d;
  printf("a = "); scanf("%d", &x);
  printf("b = "); scanf("%d",&y);
  printf("Введите тип операции: 1 - сложение; 2 - вычитание; 3 -
  умножение\n");
  scanf("%d",&d);
  printf("Результат: %d\n", (*a[d-1])(x, y));
  return 0;
}
```

Разбор примера использования указателей на функции

В приведенном примере с помощью:

```
int (*a[3])(int,int)={sum, sub, mul};
```

Задается массив указателей на три функции, каждая из которых содержит два целочисленных аргумента. Использование скобок в (*a[3]) обязательно, потому что операция * имеет более низкий приоритет, чем скобки. Соответственно, если записать так:

```
int *a[3](int,int)={sum, sub, mul};
```

то будет объявлен массив с аргументами (int,int), что противоречит синтаксису языка C.

В примере мы также одновременно инициализирует элементы этого массива физическими адресами начала кода соответствующих функций. Наконец, вызов (d-1)-й функции с фактическими значениями x, y реализуется с помощью

```
(*a[d-1])(x, y)
```

Задание 7.3

Пузырьковая сортировка с выбором направления сортировки

С использованием указателей на функции реализуйте алгоритм пузырьковой сортировки, в выбором направления сортировки. Соответствующее направление реализуйте в функциях asc (по возрастанию), desc (по убыванию). С помощью указателей на эти функции передавайте в bubble направление сортировки.

Массивы указателей

Массивы могут состоять из указателей не только на функции, но и на любые другие объекты. В этом смысле имеется возможность сохранять в массив адреса объектов разного типа данных, а также разной длины. Рассмотрим соответствующие примеры

Пример использования массива указателей на объекты разных типов

```
#include <stdio.h>
int sum(int, int);
int sum(int a, int b) {return a+b; }
main()
{
  void *a[3]={NULL, NULL, NULL};
  char s[] = "Test string";
  float n  = 3.14;
  a[0] = s; a[1] = &n; a[2] = sum;
  char *s1 = a[0];
  float *aa = a[1];
  int (*f)(int, int) = a[2];
  printf("Первый элемент массива: ");
  for (; putchar(*s1); s1++); printf("\n");
  printf("Второй элемент массива %f\n", *aa);
  printf("Третий элемент массива %d\n", (*f)(1,2));
  return 0;
}
```

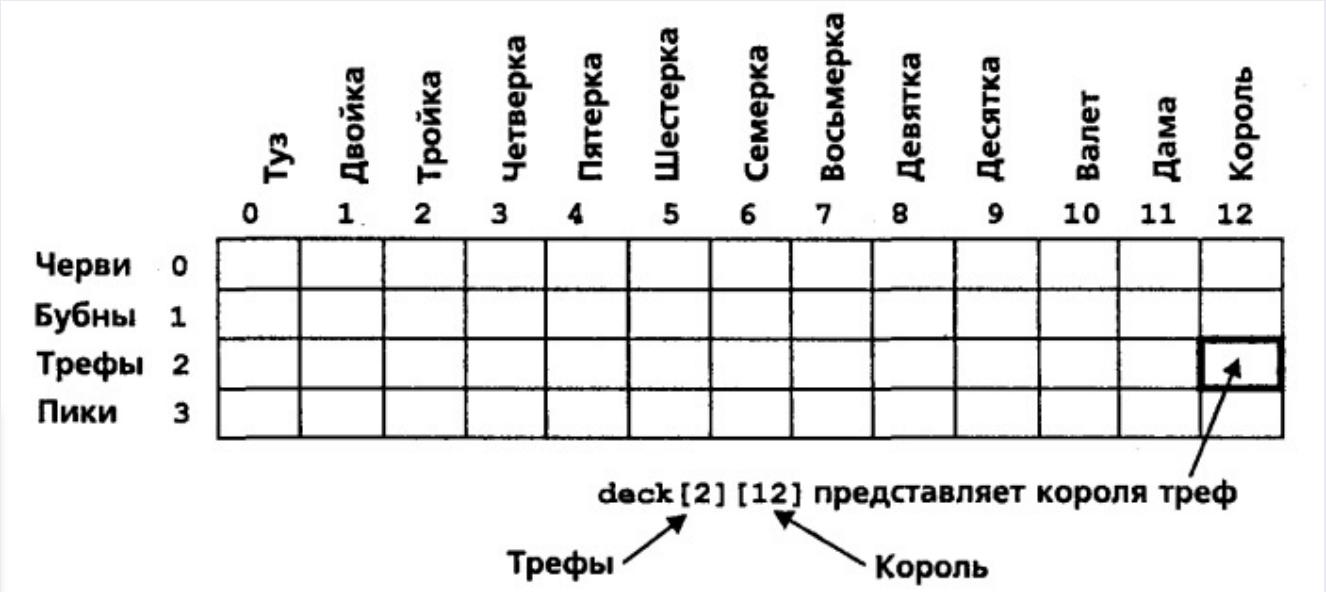
Пример использования массива указателей на данные разной длины

```
#include <stdio.h>
main()
{
  char *a[3]={"First", "Second", "Third"};
  int i;
  for (i = 0; i<3; i++)
    {
      char *s = a[i];
      for (; putchar(*s); s++);printf("\n");
    }
  return 0;
}
```

Задание 7.4

Тасование колоды карт

Представьте колоду их 52 карт двухмерным массивом и реализуйте алгоритм тасования колоды.



Задание 7.5

Прохождение лабиринта

На рисунке представлен лабиринт (единицы соответствуют стенке, нули - дорожкам лабиринта). Разработайте алгоритм прохода лабиринта. В процессе поиска выхода из лабиринта в каждый пройденный квадрат помещается символ X вместо 0. Программа должна перерисовывать лабиринт после каждого хода, так чтобы пользователь мог наблюдать процесс решения задачи.

Реализуйте разные стратегии выхода из лабиринта (указав случайную начальную позицию).

```
1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 1 0 0 0 0 0 0 1
0 0 1 0 1 0 1 1 1 1 0 1
1 1 1 0 1 0 0 0 0 1 0 1
1 0 0 0 0 1 1 1 0 1 0 0
1 1 1 1 0 1 0 1 0 1 0 1
1 0 0 1 0 1 0 1 0 1 0 1
1 1 0 1 0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 0 0 1 0 1
1 1 1 1 1 1 0 1 1 1 0 1
1 0 0 0 0 0 0 1 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1
```