ЛЕКЦИЯ 7

Указатели
Арифметика указателей
Динамическая память
Функции для работы с памятью

Указатели

Указатель – это переменная, значением которой является адрес, по которому располагаются данные.

Адрес – это номер ячейки памяти, в которой или с которой располагаются данные.

Классифицировать указатели можно:

- по типу данных (типизированные и не типизированные указатели);
- по области доступа (ближние и дальние указатели).

Указатели

Типизированный указатель – указатель, содержащий адрес данных определенного типа (системного или пользовательского).

Не типизированный указатель – указатель, содержащий адрес данных неопределенного типа (просто адрес).

Указатели

Ближний указатель – указатель, содержащий только смещение, по которому располагаются данные. Сегмент в этом случае используется по умолчанию – текущий сегмент данных. Размер ближнего указателя в 16-разрядном реальном режиме работы процессора составляет 16 бит, а в 32-разрядном защищенном режиме – 32 бита.

Дальний указатель – указатель, содержащий и сегмент и смещение. Размер дальнего указателя в 16-разрядном реальном режиме работы процессора составляет 32 бита (16 бит – сегмент, 16 бит - смещение), а в 32-разрядном защищенном режиме – 48 бит (16 бит – селектор, 32 бита - смещение).

Указатели

Работа с указателями в языке С включает три действия, осуществляемых в следующем порядке:

1. объявление указателя;
2. установка указателя;
3. обращение к значению, расположенному по указателю.

Указатели

Объявление (описание) указателя в языке С имеет следующий вид:

тип [**near**|**far**] *имя [=значение];

В современной реализации языка С (стандарт **С99**), ориентированной под разработку программ для ОС Windows, вследствие используемой в ОС Windows модели памяти, используются исключительно ближние (**near**) указатели, поэтому при объявлении указателя (например, в среде разработки Pelles C) модификатор области доступа указывать не надо.

Указатели

Указатель при объявлении можно инициализировать, указав через знак присвоения соответствующее значение. Данное значение должно быть адресом, записанном в одном из следующих виде:

- нулевое значение (идентификатор **NULL**);
- другой указатель;
- адрес переменной (через операцию взятия адреса);
- выражение, представляющее собой арифметику указателей;
- адрес, являющийся результатом выделения динамической памяти.

Указатели

Операция взятия адреса – операция языка С, возвращающая адрес переменной. Данная операция имеет следующий синтаксис:

&имя_переменной

Например, в программе описаны следующие переменные:

```
int a,b;
double c;
```

Описание указателей на эти переменные с инициализацией будет иметь вид:

```
int *ptr_a = &a, *ptr_b = &b;
double *ptr_c1 = &c, *ptr_c2 = ptr_c1;
```

Указатели

Пример объявления не типизированного указателя с инициализацией нулевым значением:

```
void *ptr = NULL;
```

Указатели

Установка указателя - присвоение его значению адреса, по которому располагаются или будут располагаться данные.

Для установки указателя используется оператор присвоения, в левой части которого указывается имя указателя, а в правой – одно из значений отличных от **NULL**, используемых при инициализации указателя. Пример установки указателей:

```
int a = 10, *ptr = NULL;
ptr = &a;
```

Указатели

Для обращения к значению, располагаемому по адресу, содержащемуся в указателе, используется операция **разыменования указателя**. Данная операция имеет следующий синтаксис:

*имя_указателя

Значение, полученное путем разыменования указателя, может рассматриваться в программе как LValue, так и RValue значение. Например:

```
double x = 0.0, *ptr = NULL;
ptr = &x;
scanf(“%lf”,ptr);
*ptr += 1.5;
printf(“%lf\n”,*ptr);
```

Указатели

В языке С можно создавать константные указатели – значение, расположенное по этому указателю нельзя изменить. Создание такого указателя имеет следующий синтаксис:

const тип *имя = инициализирующее значение;

Например, следующий фрагмент программы приведет к ошибке компиляции:

```
int a = 10;
const int *ptr = &a;
(*ptr)++;
```

Указатели и массивы

Объявление указателя на массив имеет тот же синтаксис, что и объявление обычного указателя. Например, объявление указателя на вещественный массив типа **double** будет иметь вид:

```
double *arrptr = NULL;
```

Объявление целочисленного массива из десяти элементов с инициализацией нулевыми значениями, и объявление с инициализацией указателя на этот массив будут иметь вид:

```
int arr[10] = {0}, *arrptr = arr;
```

Указатели и массивы

Фрагмент программы, в которой объявляется массив из 10 элементов целого типа, осуществляется ввод массива с вычислением суммы его элементов и вывод значения этой суммы с использованием указателей на массив и на переменную для хранения суммы:

```
int array[10] = {0}, summa = 0;
int *arrptr = array, *ptr = &summa;
for(int i=0;i<10;i++){
  scanf("%d",&arrptr[i]);
  *ptr += arrptr[i];
}
printf("Сумма: %d\n",*ptr);
```

Указатели и массивы

Довольно часто встречаются случаи, когда необходимо работать с массивами указателей. Синтаксис объявления массива указателей следующий:

тип *имя[размер];

Например: вычисление суммы набора целых чисел через обращение к ним посредством массива указателей на целые числа:

```
int arr[10], *ptrs[10], summa = 0;
for(int i=0;i<10;i++) ptrs[i] = &arr[i];
for(int i=0;i<10;i++){
  scanf("%d",ptrs[i]);
  summa+= *ptrs[i];
}
printf("Сумма: %d\n",summa);
```

Указатели и строки

Объявление указателя на строку имеет тот же синтаксис, что и объявление указателя на символьный тип данных языка С:

char *имя;

Так как в языке С строка является массивом символов, а имя массива есть указатель на этот массив, то установка указателя на строку осуществляется путем присвоения указателю имени этой строки. Например:

char str[] = “Моя строка!”, *ptr = str;

Указатели и строки

Работа со строкой как с массивом символов посредством указателя ничем не отличается от работы с массивом. Например, ниже приведен фрагмент программы вычисления длины строки *str* посредством обращения к ней через указатель *ptr*:

```
char *ptr = str;
int len = 0;
while(ptr[len]!=0) len++;
printf("Длина строки: %d\n",len);
```

Указатели и строки

Интересной является возможность объявления указателей на строки и их установка на строковые константы. Например, возможно следующее:

```
char *str = “Моя строка!”;
puts(str);
```

Указатели и строки

Например, в следующем фрагменте программы на экран выводится сообщение *“Положительное значение”*, если значение целочисленной переменной *a* больше нуля, *“Отрицательное значение”* – если меньше и *“Нулевое значение”* – если ноль:

```
int a = 0;
printf(“Введите число: ”); scanf(“%d”,&a);
char *str = NULL;
if(a > 0) str = “Положительное значение”;
  else if(a < 0) str = “Отрицательное значение”;
    else str = “Нулевое значение”;
puts(str);
```

Указатели и перечисления

Работа с указателями на перечислимый тип данных (**enum**) ничем не отличается от работы с указателями на целочисленный тип данных, так как перечислимый тип данных является производным от целочисленного типа.

Указатели и структуры

Объявление указателя на структуры или объединения, а также установка указателя на структуры и объединения синтаксически не отличается от соответствующих действий с указателями на скалярные типы данных. Например:

```
typedef struct {double x,y;} Point2D;
Point2D  pnt = {0.0,0.0}, *ptr = &pnt;
```

Указатели и структуры

Отличие заключается в обращении к полям структуры (объединения) через указатели на эти структуры (объединения). Возможны два варианта:

- (*имя_указателя).имя_поля
- имя_указателя->имя_поля

Указатели и структуры

Вычислить расстояние между двумя точками в двумерном пространстве (структура Point2D):

```
Point2D pnt[2], *ptr1 = &pnt[0], *ptr2 = &pnt[1];
printf(“Введите первую точку: ”);
scanf(“%lf %lf”,&ptr1->x,&ptr1->y);
printf(“Введите вторую точку: ”);
scanf(“%lf %lf”,&ptr2->x,&ptr2->y);
double len = sqrt(
                    pow(ptr1->x - ptr2->x, 2.0)+
                    pow(ptr1->y - ptr2->y, 2.0)
                   );
printf(“Расстояние: %lf\n”,len);
```

Указатели и структуры

Вычисление длины ломаной линии заданной массивом структур arr (структура Point2D) размера N через указатель ptr на этот массив:

```
Point2D arr[N] = {...}, *ptr = arr;
double len = 0.0;
for(int i=1;i<N;i++)
  len += sqrt(
                pow(ptr[i].x - ptr[i-1].x, 2.0)+
                pow(ptr[i].y - ptr[i-1].y, 2.0)
               );
printf("Длина ломаной линии: %lf\n",len);
```

Указатели и структуры

Вычисление длины ломаной линии заданной массивом структур *arr* (структура Point2D) размера *N* через массив указателей *ptr* на на элементы исходного массива:

```
Point2D arr[N] = {...}, *ptrs[N];
for(int i=0;i<N;i++) ptrs[i] = &arr[i];
double len = 0.0;
for(int i=1;i<N;i++)
  len += sqrt(
                pow(ptrs[i]->x - ptrs[i-1]->x, 2.0)+
                pow(ptrs[i]->y - ptrs[i-1]->y, 2.0)
               );
printf("Длина ломаной линии: %lf\n",len);
```

Арифметика указателей

В языке С доступны некоторые арифметические действия над типизированными указателями.

Доступны следующие виды выражений:

- указатель++; ++указатель;
- указатель--; --указатель;
- указатель = указатель + (целочисленное выражение);
- указатель += (целочисленное выражение);
- указатель = указатель - (целочисленное выражение);
- указатель -= (целочисленное выражение);

Арифметика указателей

Технически сложение (или вычитание) типизированного указателя и целого числа означает «сдвиг» указателя на определенное число байт (в зависимости от размера типа указателя) «вправо» (или «влево»).

Примеры:

```
int *a, *b, *c; //Объявление указателей
double *x, *y;
...                 //Установка указателей
a++;        //Сдвиг вправо на 4 байта
b-=3;       //Сдвиг влево на 12 байт
c=a+2;      //Смещение c относительно a на 8 байт
y = x--;    //X смещается влево на 8 байт
```

Арифметика указателей

Арифметика указателей наиболее часто применяется для доступа к элементам массивов. Например, вычисление суммы элементов целочисленного массива:

```
int arr[10] = {...}, *ptr = NULL, summa = 0;
ptr = arr;                                  //или ptr = &arr[0];
for(int i=0;i<10;i++,ptr++) summa +=*ptr;
printf("Сумма: %d\n", summa);
```

Арифметика указателей

Цикл в последнем фрагменте программы можно записать и несколько иначе:

```
for(int i=0;i<10;i++) summa += *(ptr+i);
```

Арифметика указателей

Еще одним вариантом арифметики указателей является вычитание указателя из другого указателя, в виде:

целочисленная переменная = указатель №1 – указатель №2;

Результатом вычитания указателей является целое значение равное расстоянию между адресами, содержащимися в указателях.

Например:

```
int arr[10], *ptr1 = arr, *ptr2 = &arr[1], *ptr3 = &arr[4];
int dest1 = ptr2 - ptr1, dest2 = ptr2 - ptr3;
printf("%d\n%d\n",dest1,dest2);
```

На экране будет выведено:

```
1
-3
```

Динамическая память

Традиционно весь объем памяти компьютера во время его работы разделяют на следующие области:

- системная область, занимаемая базовой системой ввода и вывода, операционной системой, сервисами операционной системы и драйверами различных устройств;
- область пользовательских программ, занимаемая программами или сервисами, которые запустил пользователь компьютера в процессе работы с ним;
- свободная память, доступная для загрузки других программ или сервисов.

Динамическая память

Динамическая память – это область (блок) памяти выделенный для нужд программы в процессе работы программы (а не заранее).

Основными двумя действиями над динамической памятью являются: выделение и освобождение. В языке С функции для осуществления этих действия описаны в библиотеке **stdlib.h**.

Динамическая память

Функция выделения блока памяти:

void * malloc(size_t *size*);

Например, фрагмент программы выделения динамической памяти под структуру Point2D:

```
Point2D *ptr = (Point2D *)malloc(sizeof(Point2D));
```

Динамическая память

Функция выделения блока памяти под массив:

void * calloc(size_t *num*, size_t *size*);

Например, фрагмент программы для выделения динамической памяти под целочисленный массив из 20 элементов:

```
int *array = (int *)calloc(20,sizeof(int));
```

Динамическая память

Функция изменения размера выделенного ранее блока памяти:

void * realloc(**void** **memblock*, size_t *size*);

Например, увеличение целочисленного массива до 30-ти элементов:

array = (**int** *)realloc(array,30***sizeof**(**int**));

Динамическая память

Функция освобождения динамической памяти:

void free(**void** **memblock*);

Например, освобождение блока памяти, выделенного под структуру Point2D:

free(ptr);

Динамическая память

Помимо описанных функций для работы с динамической памятью (выделение и освобождение) на практике широко используются функции работы с блоками памяти, описанные в библиотеке **string.h**

Динамическая память

Функция копирования содержимого одного блока памяти в другой блок:

```
void * memcpy(void * restrict targetbuf,
              const void * restrict sourcebuf,
              size_t num);
```

Динамическая память

Функция копирования содержимого одного блока памяти в другой блок:

```
void * memmove(void *targetbuf,
               const void *sourcebuf,
               size_t num);
```

Динамическая память

Функция сравнения двух блоков памяти:

```
int memcmp(const void *buffer1,
        const void *buffer2,
        size_t num);
```

Динамическая память

Функция заполнения блока памяти:

void * memset(**void** **buffer*, **int** *c*, size_t *num*);

Пример 1

Список точек в двумерном пространстве вводится пользователем: сначала указывается количество элементов в списке, а затем вводятся сами элементы в формате *(x,y)*. Определить две точки в списке максимально удаленные друг от друга и две точки – максимально приближенные друг к другу. Найденные точки вывести в формате: *(x,y) – (x,y) : расстояние*. Список точек создается в динамической памяти.

Пример 1

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(int argc, char *argv[])
{
  typedef struct{double x,y;} Point2D;
  int num;
  printf("Введите количество точек: "); scanf("%d",&num);
  if(num < 3) {puts("Слишком мало точек!"); return 0;}
  Point2D *list = (Point2D *)calloc(num,sizeof(Point2D));
  if(!list) {puts("Недостаточно памяти!"); return 0;}
  puts(“Введите список точек: ”);
  for(int i=0;i<num;i++){
    fflush(stdin);
    scanf("(%lf,%lf)",&list[i].x,&list[i].y);
  }
```

Пример 1

```
int mins[2] = {0,1}, maxs[2] = {0,1};
double min = sqrt(
               pow(list[0].x-list[1].x,2.0)+
               pow(list[0].y-list[1].y,2.0)
             ),
       max = min;
for(int i=0;i<num-1;i++)
  for(int j=i+1;j<num;j++){
    double len = sqrt(
                   pow(list[i].x-list[j].x,2.0)+
                   pow(list[i].y-list[j].y,2.0)
                 );
    if(len > max){
      max = len; maxs[0] = i; maxs[1] = j;
    }else if(len < min){
      min = len; mins[0] = i; mins[1] = j;
    }
  }
```

Пример 1

```
 printf("MAX: (%.2lf,%.2lf) - (%.2lf,%.2lf) : %.2lf\n",
     list[maxs[0]].x,list[maxs[0]].y,
     list[maxs[1]].x,list[maxs[1]].y,max);
   printf("MIN: (%.2lf,%.2lf) - (%.2lf,%.2lf) : %.2lf\n",
     list[mins[0]].x,list[mins[0]].y,
     list[mins[1]].x,list[mins[1]].y,min);
  free(list);
  return 0;
}
```

Пример 2

Список окружностей (координаты центра и радиус) вводится пользователем: сначала вводится количество элементов в списке, а затем – сами элементы в формате (x,y) radius. Необходимо удалить из списка все окружности, длина которых меньше средней длины. Полученный список вывести на экран.

Пример 2

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])
{
  typedef struct{double x,y,r;} CIRCLE;
  int num;
  printf("Введите количество записей: "); scanf("%d",&num);
  if(num < 2) {puts("Слишком мало записей!"); return 0;}
  CIRCLE *list = (CIRCLE *)calloc(num,sizeof(CIRCLE));
  if(!list) {puts("Few memory!"); return 0;}
  double midlen = 0;
  puts("Введите список: ");
  for(int i=0;i<num;i++){
    fflush(stdin);
    scanf("(%lf,%lf) %lf",
      &list[i].x,&list[i].y,&list[i].r);
    midlen += 2.0*list[i].r*pi;
  }
  midlen /= num;
```

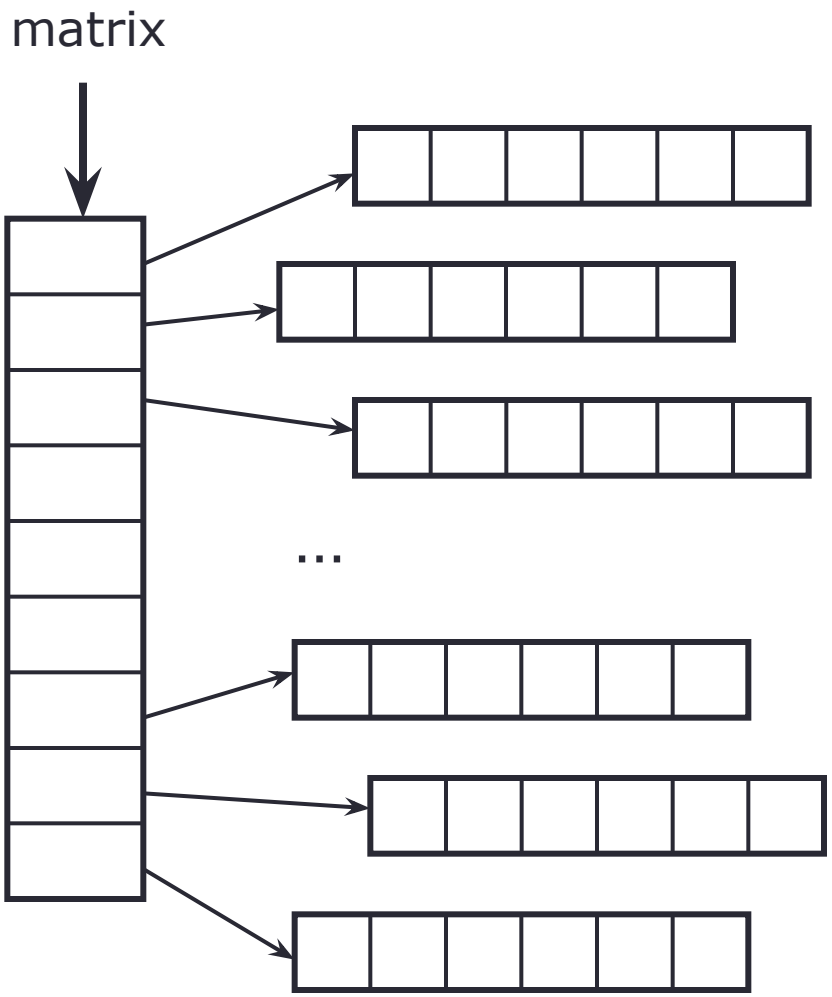
Пример 2

```
for(int i=0;i<num;i++)
    if(2.0*list[i].r*pi < midlen){
      memcpy(&list[i],&list[i+1],sizeof(CIRCLE)*(num-i-1));
      num--;
      i--;
    }
  list = (CIRCLE *)realloc(list,num);
  puts("Результат: ");
  for(int i=0;i<num;i++)
    printf("(%.2lf,%.2lf) %.2lf\n",
      list[i].x,list[i].y,list[i].r);
  free(list);
  return 0;
}
```

Пример 3

Создать в динамической памяти вещественную матрицу размера N×M (вводятся пользователем). Осуществить ввод матрицы. Упорядочить строки матрицы в порядке увеличения или уменьшения суммы их элементов (направление выбирает пользователь). Полученную матрицу вывести на экран.

Динамическая матрица



Пример 3

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])
{
  int n,m;
  printf("Введите размеры матрицы: ");
  scanf("%d %d",&n,&m);
  if((n<2)||(m<2)) {puts("Неправильный ввод!"); return 0;}
  double **matrix = (double **)calloc(n,sizeof(double *));
  if(!matrix) {puts("Мало памяти!"); return 0;}
  double *summs = (double *)calloc(n,sizeof(double));
  if(!summs) {
    free(matrix);
    puts("Мало памяти!"); return 0;
  }
```

Пример 3

```
for(int i=0;i<n;i++){
  summs[i] = 0.0;
  matrix[i] = (double*)calloc(m,sizeof(double));
  if(!matrix[i]){
    for(int j=0;j<i;j++) free(matrix[i]);
    free(matrix); free(summs);
    puts("Мало памяти!"); return 0;
  }
}

puts("Введите матрицу:");
for(int i=0;i<n;i++) for(int j=0;j<m;j++){
  scanf("%lf",&matrix[i][j]);
  summs[i] += matrix[i][j];
}
```

Пример 3

```
int type = 0;
printf("Введите 0– по возрастанию, не 0– по убыванию: ");
scanf("%d",&type);
int flag = 1;
while(flag){
  flag = 0;
  for(int i=0;i<n-1;i++)
    if((!type&&(summs[i] > summs[i+1]))||
       ( type&&(summs[i] < summs[i+1]))){
      double *ptr = matrix[i], sum = summs[i];
      matrix[i] = matrix[i+1]; summs[i] = summs[i+1];
      matrix[i+1] = ptr; summs[i+1] = sum;
      flag = 1;
  }
}
```

Пример 3

```
 puts("Результат:");
 for(int i=0;i<n;i++){
   for(int j=0;j<m;j++) printf("%6.2lf ",matrix[i][j]);
   free(matrix[i]);
   printf("\tSumma: %.3lf\n",summs[i]);
 }
 free(matrix); free(sums);
 return 0;
}
```

Пример 4

Создать в динамической памяти массив строк. Строки вводятся пользователем, признак завершения ввода – ввод пустой строки. Длина каждой строки не превышает 100 символов. Удалить из массива все строки, длина которых меньше средней длины всех введенных строк. Полученный массив вывести на экран. При реализации обеспечить эффективное хранение строк в памяти: память под строки выделяется динамически, с учетом длины строки.

Пример 4

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char *argv[])
{
  char **list = NULL;
  int count = 0, midlen = 0;
  puts(“Вводите строки”);
  while(1){
    char str[101];
    gets(str);
    if(strcmp(str,"")==0) break;
    char **tmp = (char**)realloc(list, (count+1)*sizeof(char*));
    if(!tmp) {puts("Мало памяти!"); break;}
    list = tmp;
    list[count] = (char *)malloc(strlen(str)+1);
    if(!list[count]) {puts("Мало памяти!"); break;}
    midlen += strlen(str); strcpy(list[count],str);  count++;
  }
```

Пример 4

```
  midlen /= count;
  for(int i=0;i<count;i++)
    if(strlen(list[i]) < midlen){
      free(list[i]);
      memcpy(&list[i],&list[i+1],(count-i-1)*sizeof(char*));
      count--;
      i--;
    }
  list = (char **)realloc(list,count*sizeof(char*));
  puts("Результат: ");
  for(int i=0;i<count;i++){
    puts(list[i]); free(list[i]);
  }
  free(list);
  return 0;
}
```