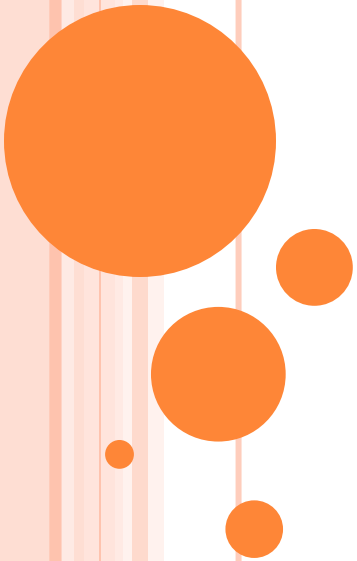


Указатели

Лекция №7

Общие сведениях об указателях

- указатель - переменная, которая содержит адрес другой переменной
- указуемая переменная – это переменная, адрес которой хранится в указателе

- указатель – аналог ярлыка в файловой системе
- указуемая переменная – аналог самого файла в файловой системе

Исходные предпосылки

- Физическая память компьютера – одномерный массив
- Необходимость гибкой работы с адресами оперативной памяти
- Си находится между Паскалем и Ассемблером
- Широкое применение Си для специализированных ЭВМ с ограниченными ресурсами
- Необходимость управления памятью – резервирование и освобождение в процессе работы программы

Формат описания указателя

- тип *имя;
  - * в описании – признак указателя
  - тип относится к указуемой переменной

- Примеры
  - int *p;
  - float *q;
  - char *s;

Свойства указателей

- освобождают от необходимости помнить адреса ячеек памяти
- поддерживают операции адресной арифметики
  - нахождение смещения одной ячейки памяти относительно другой без изменения указателей
  - перемещение от одной ячейки памяти к другой с некоторым шагом
  - сравнение указателей
  - нахождение адресного расстояния между ячейками памяти

Этапы работы с указателями

- Определение указателя
- Назначение указателю адреса переменной или элемента массива
- Работа с содержимым через указатель с применением операций адресной арифметики

- !!!После определения в указателе – случайный адрес
- Если не назначить указателю конкретный адрес – программа будет работать непредсказуемо

Инициализация указателей

- int A[5]={2,1,5,3,4};
- int *p=&A[0];//или int *p=A;

Обращение к содержимому

- a=*p
- * в выражении означает доступ к указуемому содержимому

Пример работы с указателем

- Определение указателя
- Присваивание указателю адреса другой переменной
- Работа с переменной через указатель

- int a=5, *p,b=0;
- p=&a;//обязательно присвоить адрес!!!
- b=*p;

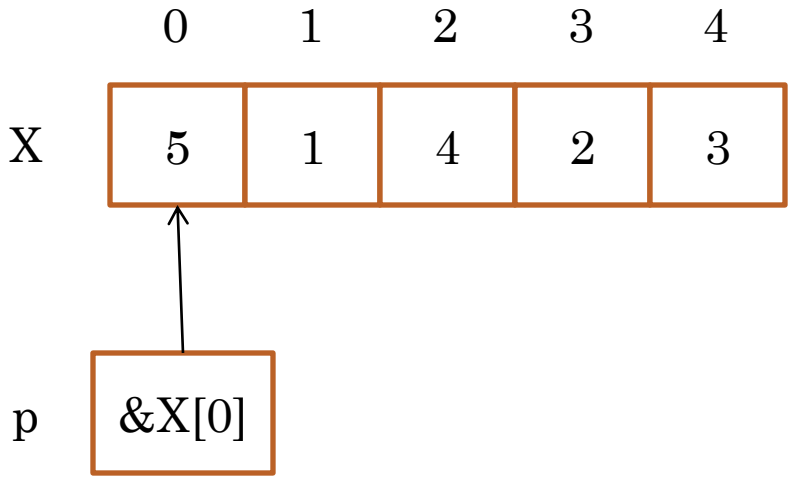
- !!! Использование указателя со случайным адресом ведет к непредсказуемым результатам

Пустой указатель

- int *t=NULL;

- Пустой указатель – нулевой адрес
- При обращении к нему формируется системное сообщение об ошибке
- Может использоваться в качестве признака пустоты, например,
  - при работе с массивами указателей, списками, деревьями
  - при возврате результата из функции

Работа с массивами с помощью указателей




```
int X[]={5,1,4,2,3}, *p, i=2, k=0;
p=&X[0];
p++;
k=*(p+i);
```

Операции адресной арифметики

- Обращение к содержимому со смещением
  - *(p+i) – содержимое ячейки на i элементов вперед
  - *(p-i) – содержимое ячейки на i элементов назад
    - смещение должно быть целым
    - размер содержимого учитывается автоматически
    - указатель не изменяется
- «Перемещение» вдоль памяти
  - p++ - на 1 элемент вперед
  - p-- - на 1 элемент назад
  - p+=i – на i элементов вперед
  - p-=i – на i элементов назад
    - указатель изменяется

Операции адресной арифметики

- Сравнение указателей
  - p<q ;адрес в указателе p ближе к началу, чем адрес в указателе q?
  - p==q; указатели указывают на одну и ту же ячейку?
- Вычитание указателей
  - k=p-q;
    - результат – целое число, показывающее, на сколько элементов адрес в указателе p дальше от начала памяти относительно адреса в указателе q
- Сложение, умножение и деление указателей не имеет смысла

Примеры операций над указателями

```
int X[]={5,1,4,2,3}, *p,*q, i=0, k=0,j=0;
p=&X[2];
q=&X[4];
p--;
k=*(p-1);//k=?
if(p<q)
   j=q-p;//j=?
if(*p>*q)
    i=(*p)-(*q);//i=?
```

Сравнение массивов и указателей

- Сходства
  - Обеспечивают доступ к элементам по номеру
  - Тип данных учитывает размер элементов
- Различия
  - Адрес массива изменить нельзя- массив привязан к конкретной области памяти
  - Адрес, записанный в указателе, можно изменить с помощью операций адресной арифметики

Эквивалентность операций над указателями и массивами

- int X[5],*p,k=2;
- p=X;

- X ~ &X[0]
- X+k ~ &X[k]
- *(X+k) ~ X[k]
- p+k ~ &p[k]
- *(p+k) ~ p[k]

- Работу с указателями можно сделать почти неотличимой от работы с массивами

Обработка массива с помощью указателя

```
int X[]={5,1,4,2,3}, *p,*q, i=0, n=5,s=0;
//Способ 1
p=X;
for(i=0;i<n;i++)
   s=s+p[i];// указатель на месте, индекс двигается
//Способ 2
for(p=X;p<X+n;p++)
   s=s+(*p);// указатель двигается
```

Поиск элемента с помощью указателей

```
int X[]={5,1,4,2,3}, *p,*q, i=0, n=5,s=0;
for(p=X;p<X+n;p++)//что делает фрагмент?
    if(*p==4)
        q=p;
s=*q;

q=X;
for(p=X+1;p<X+n;p++)//что делает фрагмент?
    if(*p<*q)
        q=p;
s=*q;
```

Перестановки элементов с помощью указателей

```
int X[]={5,1,4,2,3}, *p,*q, i=0, n=5,s=0;


p=X;
q=p+4;
s=*p;
*p=*q;
*q=s;


for(q=p;q<p+n;q++)
   printf(“%d”,*q);
```

Пример сжатия путем кодирования повторяющихся последовательностей их «координатами»

```
int X[]={5,1,4,2,3,7,1,4,2,3,9}, *p,*q, i=0, n=10,s1=0, s2=0, k=0;
int Y[6];
for(p=X;p<X+n-1;p++)//внешний цикл движения по массиву
  for(q=p+1; q<X+n; q++)//внутренний цикл движения от тек. эл.
    {while(*(p+i)==*(q+i)) //пока одинаковые элементы
          i++;//считаем их количество
      if(i>3)//Если их больше 3
             {s1=p-X;//запоминаем номер начала 1-й повт. посл.
              s2=q-X; //запоминаем номер начала 2-й повт. посл.
             Y[k]=i;//сохраняем длину - ключ
             for(k=1;k<i;k++)
                Y[k]=X[s1+k];//сохр. саму послед.
             Y[k++]=s1; //сохраняем коорд. начала
             Y[k++]=s2;}
      }
```

Выводы

- Указатели позволяют гибко работать с оперативной памятью
- Указатель позволяет работать с памятью как с массивом с помощью адресной арифметики
- Указатель позволяет «перемещаться» вдоль памяти, массив жестко привязан к участку памяти
- Тип данных при работе с указателями учитывается автоматически
- Указатели позволяют динамически работать с памятью во время работы программы