Динамические переменные и массивы

Лекция №8

Общие сведениях о динамических переменных

- Динамическая переменная – это переменная, которая появляется в памяти и исчезает из памяти в процессе работы программы
- Обычные переменные появляются в памяти до начала работы программы, и исчезают из памяти перед окончанием работы программы

Исходные предпосылки

- Необходимость хранения только тех данных, которые нужны в данный момент
- Необходимость гибкого управления размерами массивов. (у обычных массивов размер фиксированный)
- Необходимость организации сложных структур данных, в которых элементы связаны через указатели. К ним относятся массивы указателей, списки, двоичные деревья, графы

Правила работы с динамической памятью

- Для работы с динамической памятью используются специальные команды: new и delete
- Динамическая память доступна только через указатель
- Динамическая память должна быть освобождена до окончания работы программы
- Должна быть предусмотрена проверка успешности захвата памяти
- Указатель на динамическую память нельзя терять
- Нельзя дважды освобождать одну и ту же память
- Нельзя освобождать переменную, которая не является динамической

Форматы команд

- Захват памяти
- указатель= new тип [размер];
  - размер – целая переменная или выражение

- Освобождение памяти
- delete указатель;

Пример работы с динамическими переменными

```
#include<stdio.h>
void main(void)
{
int *p;
p=new int;
if(p==NULL)
 {printf(“No memory”);
   return;
}
*p=3;
printf(“%d”,*p);
delete p;
```

Пример работы с динамическим массивом

```
#include<stdio.h>
void main (void)
{int *p,*q, n, i;
printf(“array size ?”);
scanf(“%d”, &n);
p=new int[n];
If(p==NULL)
    {return;
     printf(“No memory”);}
for(i=0; i<n; i++)//ввод
   {printf(“Element %d”,i);
     scanf(“%d”, p+i);}
for(q=p; q<p+n; q++)//вывод
    printf(“%d\n”,*q);
delete p;
}
```

Добавление элемента в динамический массив

```
int *p,*q,*t,n=5,i;
p=new int[n];//создание исходного массива
for(i=0;i<n; i++)//заполнение исходного массива
  *(p+i)=i;
q=new int [n+1];//создание расширенного массива
for(i=0;i<n; i++) //копирование в расширенный массив
   *(q+i)=*(p+i);
delete p;   //удаление исходного массива
*(q+i)=i;   //заполнение дополнительного элемента
n++;  //корректировка количества элементов
for(t=q; t<p+n; t++)//вывод расширенного массива
  printf("%d",*t);
```

Удаление элемента из динамического массива

```
int *p,*q,*t,n=5,i;
p=new int[n];//создание исходного массива
for(i=0;i<n; i++)//заполнение исходного массива
  *(p+i)=i;
q=new int [n-1];//создание укороченного массива
for(i=0;i<n-1;i++) //копирование в укороченный
   массив
   *(q+i)=*(p+i);
delete p;   //удаление исходного массива
n--;  //корректировка количества элементов
for(t=q; t<p+n; t++)//вывод укороченного массива
  printf("%d",*t);
```

Динамические строки

```
char s1[]="This is a string!",*s2,*s3,*s4,k=0;
s2=s1;//запоминание адреса строки s1
while(s2[k]!=' ')
  k++;
s3=new char[k+1];
for(s4=s3,s2=s1;s4<s3+k;s4++,s2++)
   *s4=*s2;
*s4='\0';
printf("%s",s4);
```

Указатели на указатели

- Указатель на указатель – это указатель, который содержит указатель на переменную (адрес адреса)
  - Полезны для управления размером массива
  - Полезны для организации сложных структур данных
  - Полезны для работы с двумерными массивами и массивами указателей

- Формат указателя на указатель
- тип **имя;

- int **p;

Интерпретации указателя на указатель

- Указатель на указатель на одиночную переменную
- Указатель на указатель на массив

Операции с указателями на указатели на переменные

```
int **pp,*p, a=4,*q, b;
p=&a;
pp=&p;

q=*pp;//сохранение указателя, на который
        //указывает pp
b=**pp;//сохранение содержимого указателя,
        //на который указывает pp
```

Добавление элемента в динамический массив

```
int  **pp, *p,*q,*t,n=5,i;
p=new int[n];//создание исходного массива
*pp=p;//сохранение адреса
for(i=0;i<n; i++)//заполнение исходного массива
  **(pp+i)=i;
q=new int [n+1];//создание расширенного массива
for(i=0;i<n;i++) //копирование в расширенный массив
   *(q+i)=*(p+i);
*(q+i)=i;   //заполнение дополнительного элемента
n++;  //корректировка количества элементов
*pp=q;
delete p;   //удаление исходного массива
for(t=*pp;t<*pp+n;*pp++)//вывод расширенного массива
  printf("%d",*t);
delete q; delete pp;
```

Массивы указателей

- Массив указателей – это массив, элементом которого являются адреса переменных или массивов

Преимущества массивов указателей

- Позволяют переставлять адреса элементов, а не сами элементы
- Позволяют упорядочивать не только одиночные переменные, но и более сложные объекты, например, строки

- Дают возможность создать пользовательское меню

Формат массива указателей

- тип *имя[размер];

- double *pd[5];
- char *ps[3];

- Указатель на массив и массив указателей – разные вещи

Интерпретации массивов указателей

- массив указателей на одиночные переменные
- массив указателей на массивы

Варианты формирования массивов указателей

- Массив указателей – обычный, элементы – обычные
- Массив указателей – обычный, элементы – динамические
- Массив указателей динамический, элементы обычные
- Массив указателей динамический, элементы динамические

Способы задания массивов указателей

- Инициализация элементов
  - double a=2,b=5,c=4;
  - double *pd[3]={&a,&b,&c};
- Заполнение указателями в процессе работы
  - double *pd[3],*p; int i;
  - for(i=0;i<3;i++)
    - {p=new int;
    - scanf(“%f”,p);
    - double pd[i]=p;}

Пример упорядочения массива указателей

```
double a=4,b=3,c=5,d=1,e=2;
double *pd[5]={&a,&b,&c,&d,&e},*tmp;
int i,j,n=5;


for(i=0;i<n-1;i++)
  for(j=1;j<n;j++)
   if(*pd[i]>*pd[i+1])//сравниваются содержимые
          {tmp=pd[i];//переставляются адреса
            pd[i]=pd[i+1];
            pd[i+1]=pd[i];}
```

Поиск самой длинной строки

```
char* text[]={“table”,”chair”,”bed”};
int i,k=0,kmax=0;
kmax=strlen(text[0]);
for(i=1;i<3;i++)
   {k=strlen(text[i]);
     if(k>kmax)
        kmax=k;
   }
printf(“%s”, text[kmax]);
```

Выводы

- Динамическое управление памятью позволяет обрабатывать большие объемы данных
- Команды для управления динамической памяти стали частью стандарта языка Си
- При работе с динамической памятью необходимо соблюдать ряд правил