Лекция № 19

Динамические массивы

Динамические массивы

```
int a[100]; // память на этапе компиляции
```

Если требуется хранить массивы
10, 1000, 1000, 1000 000 объектов
в разные моменты работы программы ?

Массив, в котором можно изменять количество элементов во время работы программы называется динамическим или размещаемым.

Функции C malloc и free

```
#include <stdlib.h>
```

```
void *malloc(size)
```

Функция malloc возвращает адрес на первый байт области памяти размером size байт.

Важно выполнять проверку, что возвращаемое значение не равно NULL.

Функция free() освобождает место в памяти.

Выделение памяти под переменную

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct student
{
  char fio[64];
  int  money;
};
typedef struct student student;
int main(void)
{
  student *sp;
  sp = (student *) malloc (sizeof(student));
  sp->money=9000; strcpy(sp->fio,"Petrov S.C.");
  printf("Print....%d\t%s", sp->money,sp->fio);
  getch(); free(sp); return 0;
}
```

Использование malloc

```
sp = (student *) malloc (sizeof(student));
```

Делаем преобразование **void*** к **student***.

Применяем **sizeof** для вычисления занимаемой памяти в байтах.

Делаем проверку на NULL.

```
if (sp == NULL)
  {
    printf("ERRRRRRRR");
    exit(1);
  }
```

Массивы и указатели

При определении массива автоматически определяется указатель на нулевой элемент.

```
int a[100]; // массив
    // и как бонус указатель a
    // на нулевой элемент

int *p;

p=a;      // ----- одинаково
p=&a[0]; // ----------------

a[10]=34;   // ----- два способа
*(a+10)=34;
```

Выделение памяти под массив 1D

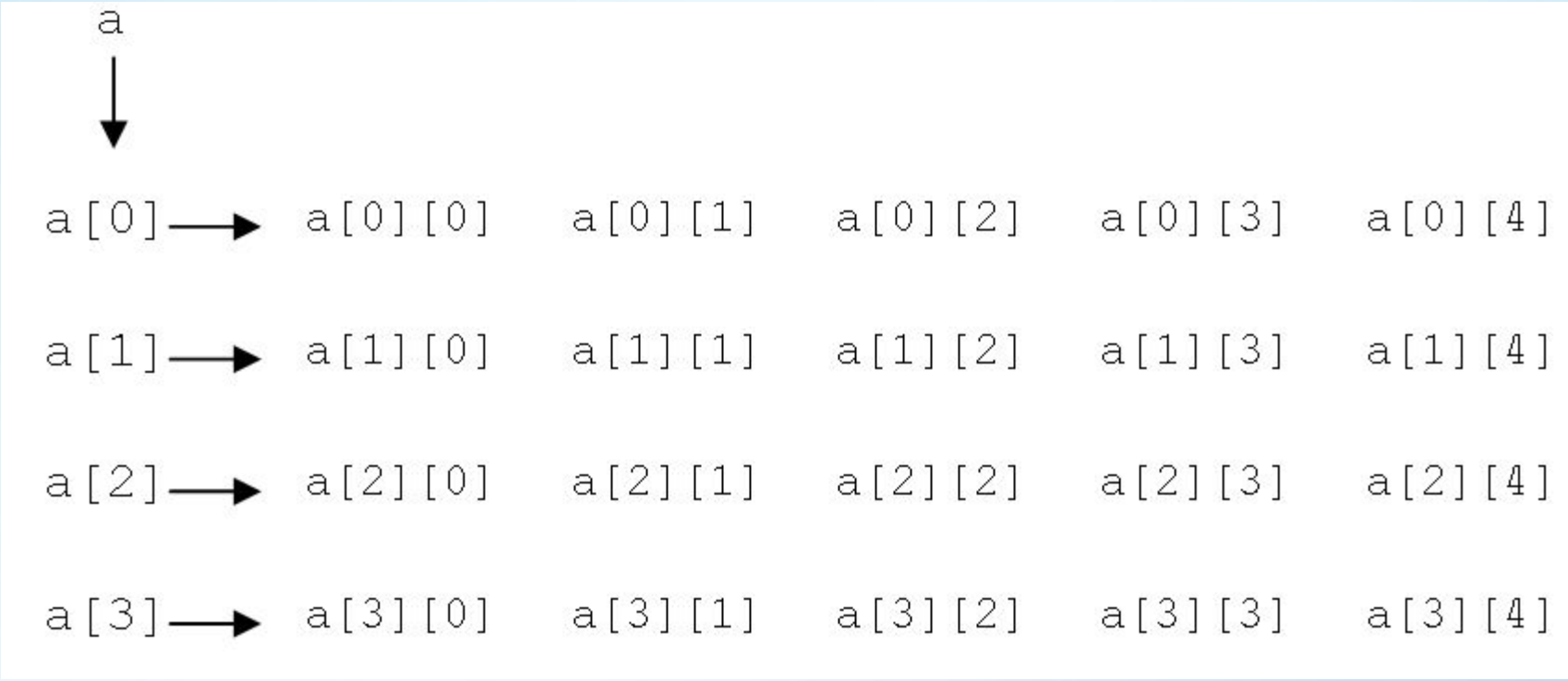
```
#include <stdio.h>
#include <conio.h>
int main(void)
{
  int k, *p;
  p = (int *) malloc (10*sizeof(int));
  // выделение памяти на 10 элементов

  for (k=0; k<10; k++)
  {
    p[k]=k;
    printf("%4d",p[k]);
  }
  free(p); // освободили память
  getch();
}
```

Хранение 2D массива

Двумерный массив
Одномерный массив, у которого элементы одномерные массивы.

Распределение памяти под массив **a[4][5]**.



Выделение памяти под массив 2D

Одномерный массив
"Указатель на целый тип"

```
int *p;
```

Двумерный массив
"Указатель на Указатель на целый тип"

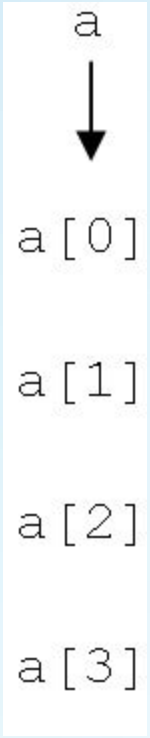
```
int **p;
```

1. Создаём массив указателей (строки).
2. С каждым указателем связываем одномерный массив (столбцы).

Выделение памяти под массив 2D

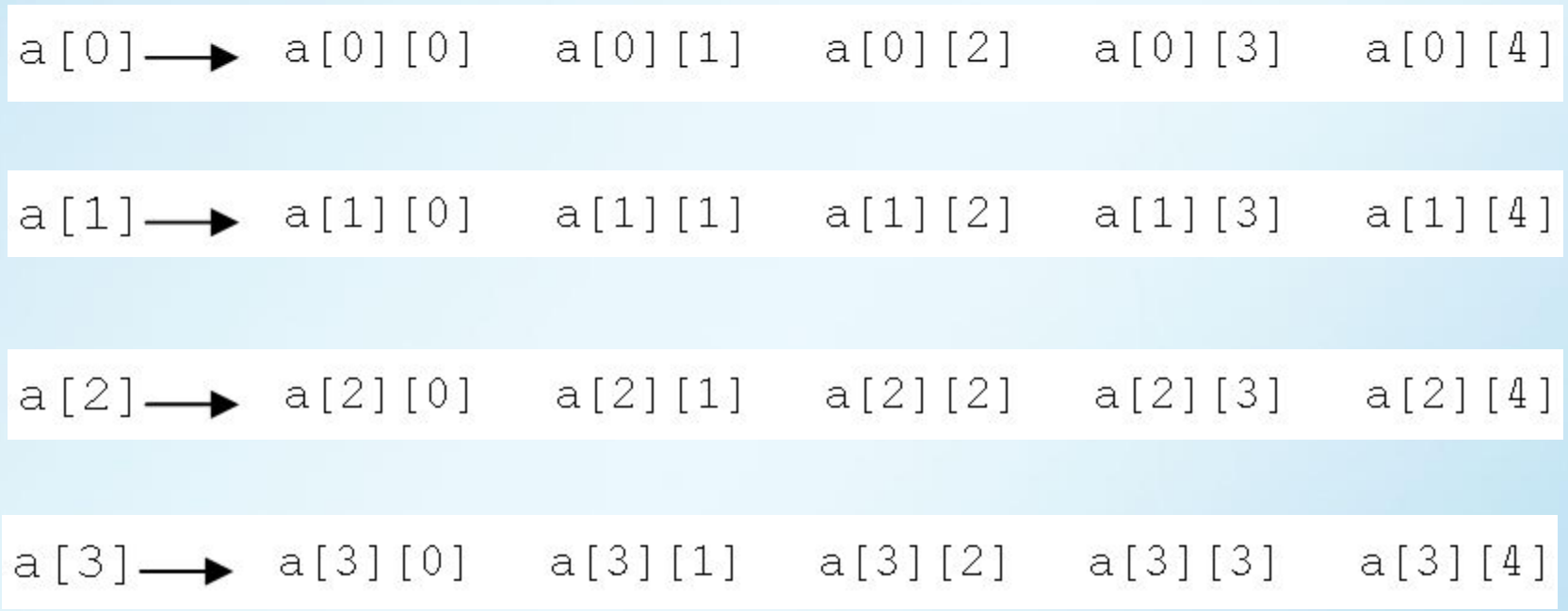
```
int **a, i, j, Mi=3, Mj=8;

a = (int **) malloc (Mi*sizeof(int*));
// указатель на массив указателей
```



Выделение памяти под массив 2D

```
for (i=0; i<=Mi-1; i++)
a[i]=(int *) malloc (Mj*sizeof(int));
```



Пример массив 2D

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
  int **p, i, j, Mi=3, Mj=8;

  // указатель на массив указателей
  p = (int **) malloc (Mi*sizeof(int*));

  // массив указателей на одномерные массивы
  for (i=0; i<=Mi-1; i++)
  p[i]=(int *) malloc (Mj*sizeof(int));
```

Пример массив 2D

```
 for (i=0; i<=Mi-1; i++)
 {
  for (j=0; j<=Mj-1; j++)
  {
    p[i][j]=rand()%10;
    printf("%3d",p[i][j]);
  }
  printf("\n");
 }
 getch();

 for (i=0; i<=Mi-1; i++) free(p[i]);
 // освобождаем память

 free(p);
}
```

Операторы C++ new и delete

Оператор **new** распределяет память во время выполнения.
Указывается количество байтов памяти, которое требуется программе.

Возвращает указатель на начало области этой памяти.

Выделение символьного массива.

```
char *buffer = new char[50];
```

При ошибке возвращает NULL-указатель.

Оператор **delete** освобождает память

Перегрузка delete

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void operator delete(void *pointer)
{
    char *data = (char *) pointer;
    int i;
    for (i = 0; i < 100; i++) data[i] = 0;
    printf("CLEANING.......");
    free(pointer);
}
```

Особая работа оператора delete !

```
int main(void)
{
    char *pointer = new char[100];
    strcpy(pointer, "SECRETS-------------");
    delete pointer;
    return 0;
}
```

Выделение памяти под объекты

```
#include <stdio.h>
#include <conio.h>

class point
{
  int x, y, z;
  public:
  point(int a, int b, int c) {x=a; y=b; z=c;}
  ~point() {printf("Destructing\n");}
  void show() {printf("%d \t%d \t%d",x,y,z);}
};

int main(void)
{
  point *p;
  p = new point(5, 6, 7);
  p->show();
  getch(); delete p; return 0;
}
```

Выделение памяти под массив объектов

```
#include <stdio.h>
#include <conio.h>

class point
{
  int x, y, z;
  public:
    point(int a, int b, int c) { x=a; y=b; z=c; }
    point() {}
    void show() {printf("%d \t%d \t%d",x,y,z);}
    void set(int a, int b, int c) { x=a; y=b; z=c; }
};
```

В классе **point** два конструктора.

Так как динамический массив не может быть инициализирован, то требуется конструктор без параметров.

Выделение памяти под массив объектов

```
int main()
{
  point *p;
  int i;

  p = new point[10];

  if(!p) { printf("ERRRRR"); return -100; }

  for(i=0; i<10; i++)  p[i].set(1, 2, 3);
  for(i=0; i<10; i++)  p[i].show();

  getch();
  delete p;
  return 0;
}
```