Динамическое распределение памяти

Время жизни динамически размещаемых данных определяется самой программой в процессе её выполнения, в отличие от статически определённых данных, чьё время жизни задаётся заранее.
Если программе требуется дополнительная память для хранения данных, она может запросить её во время работы, используя стандартные средства языка программирования.
Когда данные больше не нужны, область памяти, выделенная под них, может быть освобождена.

Таким образом, распределение памяти возложено на программу (или программиста), что даёт большую гибкость, но одновременно требует внимательности и аккуратности.

Особенности динамически размещаемых данных:

1. В отличие от заранее определённых данных, они **не имеют имён**, и доступ к ним осуществляется только через указатели.
2. Память под них выделяется **во время выполнения программы**, а не на этапе компиляции.
3. Благодаря этому, память можно как выделять, так и освобождать непосредственно в процессе работы программы, подстраивая использование ресурсов под реальные потребности.

Средства языка C для динамического распределения памяти

Для работы с динамически распределяемой памятью в языке C существует ряд функций, описанных в библиотеке **`stdlib.h`**. Чтобы использовать их, необходимо подключить эту библиотеку:

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

Преобразование указателя типа `void` в языке C

Тип **`void`** является особым: в отличие от других базовых типов данных, он не имеет ни набора допустимых значений, ни набора операций. Поэтому невозможно описывать обычные переменные типа `void`, так как им нельзя присвоить значение.

Тем не менее, `void` введён для выполнения специальных функций:

1. Указание того, что функция **не возвращает** значения:

   ```
   void FuncName() { ... }
   ```

2. Указание того, что функция **не принимает аргументов**:

   ```
   int FuncName(void) { ... }
   ```

3. Объявление указателя на тип `void`:

   ```
   void *ptr;
   ```

   Такой указатель является универсальным — он может указывать на данные любого типа. Однако для работы с ними требуется явное преобразование (`casting`) к нужному типу:

   ```
   int *p = (int *)malloc(sizeof(int));
   ```

```
int *px;
float *py;
int x = 1;
float y = 1.6;

px = &x;
py = &y;

void *z; //для того чтобы можно было работать с *z* мы должны его преобразовать
// величина типа указатель на void может быть преобразована к величине льбого другого указательного типа
int *px; px = &x;
float *py; py = &y;
void *z; (int *)z = px; (float *)z = py;

// наш указатель z указывает не на адрес конкретного объекта программы, а указывает на адрес памяти по которому может храниться любой объект некоторого типа
//это для того чтобы не писать множество однотипных ф-й возвращающих указатели на разные типы данных, и эта воз-ть используется и в динамическом распределен

// очень часто ф-ии возвращают указатель типа void

int main (){
    ...
    int *px;
    float *py;
    ...
    px = (int *)F(...);
    py = (float *)F(...);

    return 0;
}

void *F(...){...}
//возвращает чисто адрес (типо одной ячейки), а там уже расширяем количество ячеек для указателя
```

Ф-ии языка C для создания динамических данных

1. `malloc` - функция выделяет блок памяти размером как минимум `size` байт.
   В качестве своего значения функция возвращает указатель на выделенную область памяти, начинающуюся на границе слова.
   При выделении блока памяти происходит его выравнивание для размещения объектов любого типа.
   При этом инициализация блока памяти нулями не производится.

```
void *malloc(unsigned int size); //заголовок
                      // ^- это объем памяти который необходимо
```

2. `calloc` - динамически выделяет область памяти nelem (количество элементов) по elsize памяти и возвращает указатель на первый элемент массива, также как для ф-ии malloc происходит его выравнивание на границу слова.
   Отличие от malloc: происходит авто обнуление выделяемой памяти

```
//                          колво этементов которому нужно выделить
//                          |                 память на каждую переменную
//                          V                  V
void *calloc(unsigned int nelem, unsigned int elsize);
```

3. `realloc` - функция изменения размера памяти под динамические данные