**Void Pointers**

Pointer to int or (int \*) cat hold the address of variable of type int only. It would be incorrect if we assign an address of a float variable to a pointer of type pointer to int.

Void pointer can point to a variable of any daya type

Syntax:

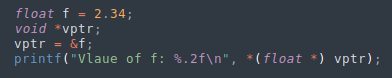


**Dereferensing a void pointer**

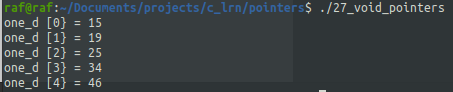
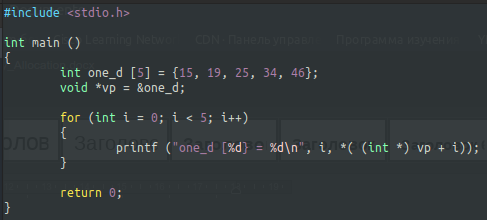
We can’t just dereference a void pointer using indirection (\*) operator. Before you dereference a void pointer it must be typecasted to appropriate pointer type



**Note**: typecasting changes type of **vptr** temporarily until the evaluation of the expression, everywhere else in the program **vptr** is *still a void pointer.*



Before apply pointer arithmetic in void pointers make sure to provide a proper typecast first:



**malloc ()**

It is used for allocate memory at runtime.

Syntax:



This function accepts a single argument called **size** of type **size\_t.**

It <stdlib.h> is defined is unsigned int.

If successfull, **malloc ()** returns a ***void pointer***to the first allocated byte of memory.

Before use – need to cast it to appropriate type



**p** is a pointer of type (datatype \*) and size memory space in bytes you want to allocate.



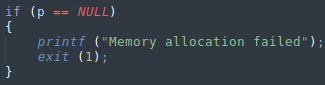
Allocates 20 contiguous bytes of memory from the heap and assingn the address of the first byte to **p**.

Memory allocated contains garbage.



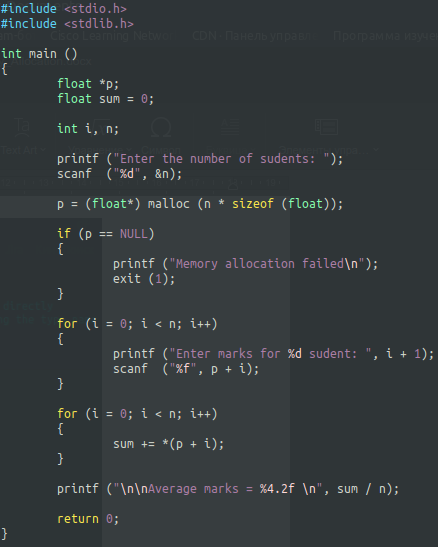
When the heap runs out of free space **malloc ()** returns **NULL**.

Before using the pointer variable we must fist always check the value returned by malloc ()



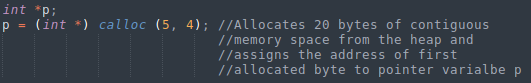
To use malloc () we need to include stdlib.h

It’s good practice to free (p) after use malloc ().



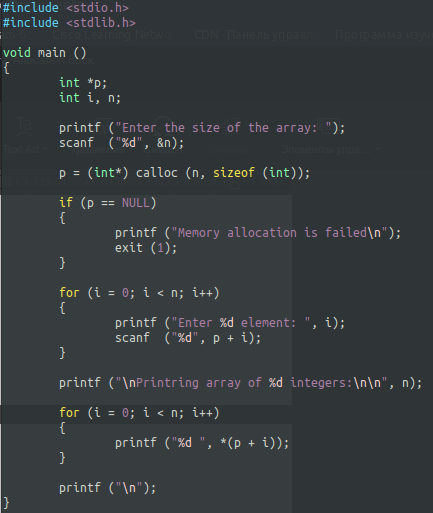
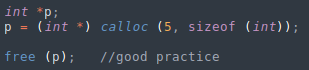
**calloc ()**

Syntax:



malloc () – allocated memory contains garbage

calloc () – allocated memory contains 0 (initialized to 0)



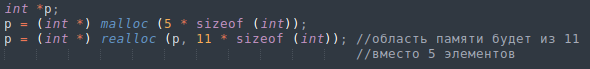
**realloc ()**

Syntax:



\*ptr – is a pointer to the fires byte of memory that was previosly allocated using malloc () / calloc ()

newsize – new size of the block in bytes



Если доступно достаточно памяти, то функция realloc () выделяет требуемое количество байт рядом с уже использованными байтами. В этом случае адрес памяти, на которую указывает **p** не меняется. Старые данные не меняются, не теряются.

Если места недостаточно:

* realloc () распределяет нужную область в другом месте кучи (heap)
* Данные копируются
* Адрес **p** меняется

It’s good practice to free (p) after use malloc ().