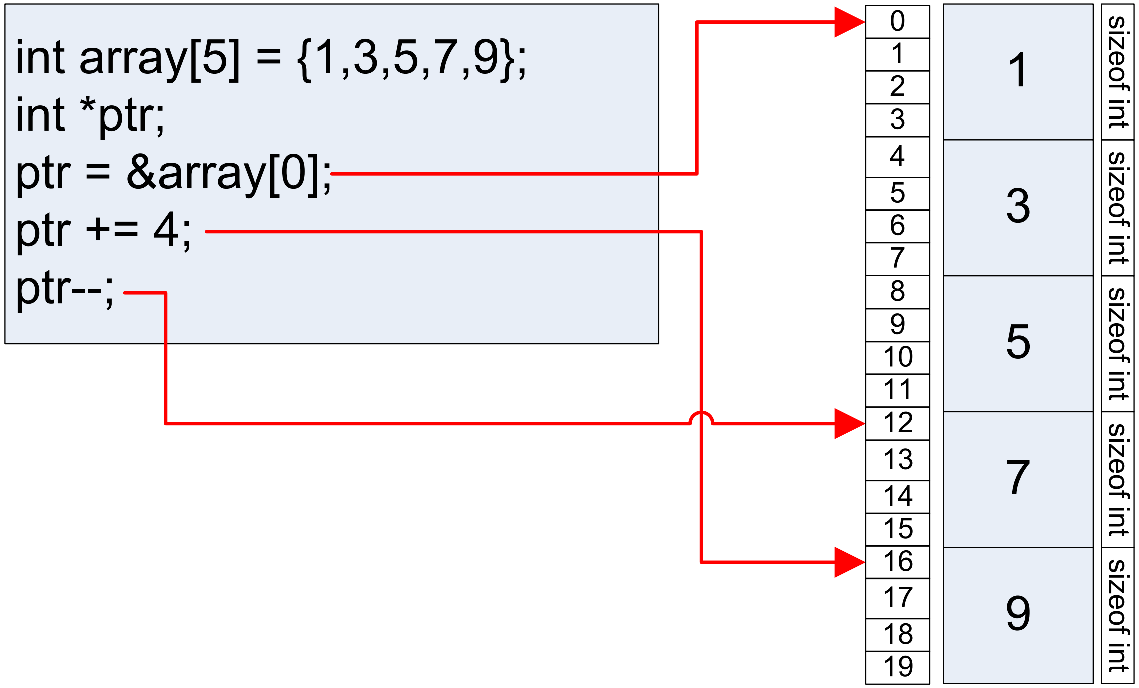
sizeof operator

- Unary operator that calculates a type size and expresses it in bytes (Remind yourself what byte is)
- Works with expression, but also with type
  - When applied to expression, it returns size of the expression’s type
  - When applied to type, it returns its size
- By definition sizeof(char) is always 1

```
int* ptr;
size_t s;

s = sizeof(*ptr);
printf(“%d\n”, s);

s = sizeof(int);
printf(“%d\n”, s);

s = sizeof(ptr);
printf(“%d\n”, s);
```

Pointers and operations on them 1/4

- Addition and subtraction of integer:
  - `data_type* ptr;`
    `ptr ± n <=> ptr ± n * sizeof(data_type)`
  - Same goes for unary operators ++/--



```
int array[5] = {1,3,5,7,9};
int *ptr;
ptr = &array[0];
ptr += 4;
ptr--;
```

Pointers and operations on them 2/4

- Subtracting two pointers – only if they are of the same type

```
#include <stddef.h>
int array[5] = {1,3,5,7,9};
int* ptr1;
int* ptr2;
ptrdiff_t diff;

ptr1 = &array[1];
ptr2 = &array[4];
diff = ptr2 - ptr1;
```

diff = 3

- The results will be defined only if the pointers point to parts of the same memory block

```
int array1[5] = {1,3,5,7,9};
int array2[5] = {2,4,6,8,10};
int* ptr1;
int* ptr2;
ptrdiff_t diff;

ptr1 = &array1[1];
ptr2 = &array2[4];
diff = ptr2 - ptr1;
```

Undefined result

Addition of two pointers is not allowed

Pointers and operations on them 3/4

- Comparing pointers:
  - It is possible to compare only object pointers

```
int array[5] = {9,7,5,3,1};
int* ptr1;
int* ptr2;

ptr1 = &array[1];
ptr2 = &array[4];
if(ptr1 < ptr2)
  printf(“Expected\n");
else
  printf(“Unexpected\n");
```

```
Output: Expected
```

- Again, the results will be defined only if the pointers point to parts of the same memory block

Pointers and operations on them 4/4

- Operator []

```
def: A[B] <=> *(A + B)
```

A + B has to be of pointer type because unary * works only with pointers

Therefore, either A has to be a pointer, and B an integer, or the other way around!

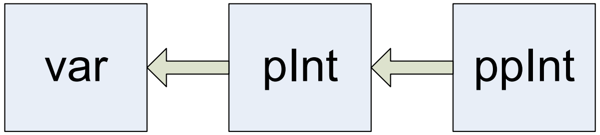
```
data_type* A; int B;
A[B] <=> *(A + B) <=> *(A + B * sizeof(data_type))
data_type* A; int B;
B[A] <=> *(B + A) <=> *(A + B * sizeof(data_type))
```

```
float* p;
float x;

/* let p be 1000, i.e. let p point to address 1000 */

x = *p; // x is float value on address 1000
x = p[0]; // x is float value on address 1000
x = p[4]; // x is float value on address 1000 + 4*sizeof(float)
x = 4[p]; // x is float value on address 1000 + 4*sizeof(float)
```

Pointer on pointer 1/2

- Pointer can point to any other type, and therefore it can point to another pointer type

```
int var;
int* pInt = &var;
int** ppInt = &pInt;
```



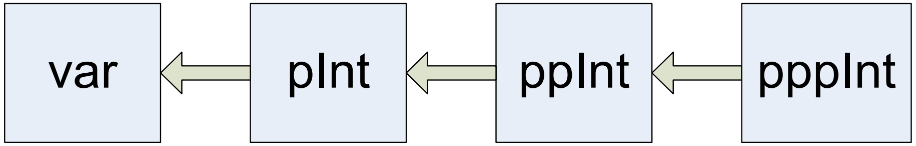
var

pInt

ppInt

- And so it goes...

```
int var;
int* pInt = &var;
int** ppInt = &pInt;
int*** pppInt = &ppInt;
```



var

pInt

ppInt

pppInt

Pointer on pointer 2/2

- When do we need that?
- 1. Passing pointers by reference

```
int g_var;

void bar(int** p)
{
   /* change pointer value */
   *p = &g_var;

   /* change value of variable to
    which pointer points to */
   **p = 39;
}
```

```
void foo()
{
   int var;
   int* ptr= &var;
   bar(&ptr);
   ...

}
```

- 2. Multidimensional arrays...

Function pointers

- Here is how to declare it:

```
return_type (*name)(param_type, param_type);
```

- Similar to arrays, function names can be reduced to pointer

```
char* (*fptr) (char* to, const char* from);

fptr = strcpy;  /* OK */
fptr = &strcpy; /* OK */
```

- Calling a function through pointer

```
char src[128];
char dst[128];

fptr(dst, src);    /* OK */
(*fptr)(dst, src); /* OK */
```

Pointer to void

- Why is pointed-to type important in pointer declaration?

```
int* p;

*p
```

- Because C is statically typed language, and this expression has to have a type. (It is similar with addition of integers and pointers).

- Pointer on some type can take only address of object of that same type, otherwise compiler will report warning or error.
- But that is not true for void pointer.

```
int var;
float* fptr;
void* vptr;

fptr = &var; /* compiler warning */
vptr = &var; /* OK */
```

Pointer to void

- Pointers to void are sometimes needed for circumventing constraints imposed by statically typed nature of C

```
void* malloc(size_t size);


int*    i = malloc(4);
double* d = malloc(8);
```

```
void qsort(void* ptr, size_t count, size_t size,
    int (*comp)(const void*, const void*));

int compare_ints(const void* a, const void* b) {
    int arg1 = *(const int*)a;
    int arg2 = *(const int*)b;

    if (arg1 < arg2) return -1;
    if (arg1 > arg2) return 1;
    return 0;
}

qsort(ints, size, sizeof(int), compare_ints);
```

```
pthread_create(...,func1,(void*)&args1);
pthread_create(...,func2,(void*)&args2);

struct params1
{
  int handle;
  int value;
} args1 = {0x45689216, 57};

struct params2
{
  short id;
  char* ident;
} args2 = {17, "hd0"};
```

```
void* func1(void* param)
{
  struct params1* args;
  args = (struct params1*)param;
  printf("%x %d", args->handle, args->value);
}


void* func2(void* param)
{
  struct params2* args;
  args = (struct params2*)param;
  printf("%d %s", args->id, args->ident);
}
```

Pointers and const qualifier 1/2

- Pointer can be modified, but not that to which it points to.
- Keyword **const** has to be left of ‘*****’

```
const type* ptr_variable;
type const* ptr_variable;
```

```
int var1 = 3;
int var2 = 5;
const int* ptr = &var1;
ptr = var2; /* OK */
*ptr = 7;   /* error */
```

- When we want to assign value of such pointer to normal, non-const pointer.

```
int* ptr;
int const* cptr;

ptr = cptr;  /* compiler warning or error */

ptr = (int*)cptr; /* OK */
```

Pointers and const qualifier 2/2

- It is possible to modify that to which pointer is point to, but not the pointer itself
- Keyword **const** has to be left of ‘*’

```
type* const ptr_variable;
```

```
int var1 = 3;
int var2 = 5;
int* const ptr = &var1;
ptr = var2; /* error */
*ptr = 7;   /* OK */
```

- Double **const** pointer is also possible

```
const type* const ptr_variable;
type const* const ptr_variable;
```

```
int var1 = 3;
int var2 = 5;
int const* const ptr = &var1;
ptr = var2; /* error */
*ptr = 7;   /* error */
```