ARRAY.

- Window into memory.
- C arrays tell you where to find data ; $\underbrace{\text{How much data;}}_{\text{not always.}}$ What type the data has.

Declare array.

INTEGER CONSTANT EXPR.

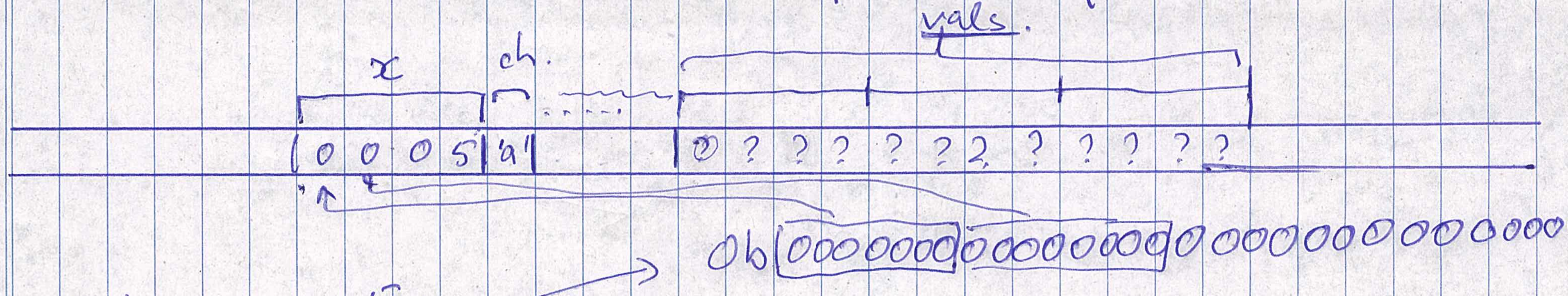
TYPE SPEC IDENT [........]

Element type.
Base type.

Name.
valid c variable name.

Capacity of the array.
how many elements can the array hold

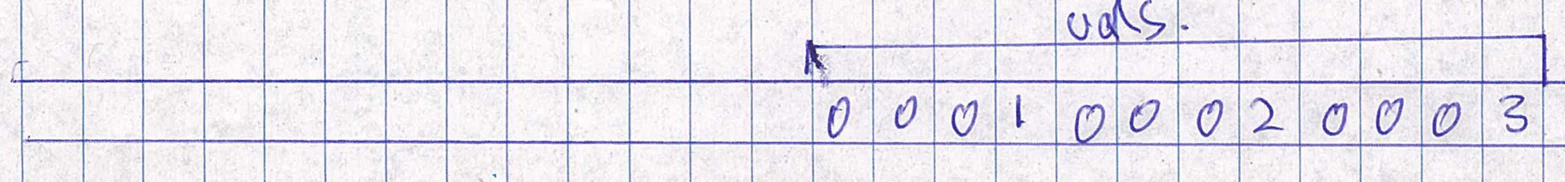
Low level visualisation of memory.


vals.
x
ch.
0 0 0 5 'a' 0 ? ? ? ? ? ? ? ? ? ? ?
0b00000000000000000000000000000101

int x = 5;
char ch = 'a';
~~8~~ int vals[3]; ← Uninitialised array → undefined values in all elements.

int vals[3] = { 0 }; → Fills 3 ints with 0.
= { 1 }; → First element is assigned 1
Other two are 0.

int vals[ ] = { 1, 2, 3 };



Using elements of array:

To insert 6 in first element of vals:

```
vals[0] = 6;
```

behaves likea a single variable of type int.

To find out what is the second element

```
if ( vals[1] == 9 ) { .... }.
```

```
int val0 , val1 , val2 ;
```

Indexing into an array with a variable

Dereferencing

size_t.

long long int

```
int i = 0;
while ( i ≤ 3 ) {
    scanf( "%d", &vals[i] );
~~}~~
    i++;
}
```

sending the address of an int to scanf.

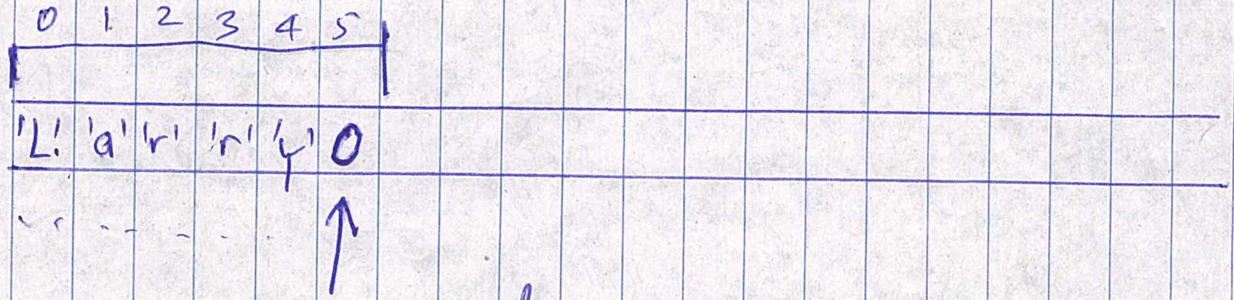
vals[i] – the item at position 'i+1'

int.

4.4

C has no proper string type. We have arrays of __char__.

char me = "Larry";



Numeric value
__ZERO.__

marks the end of the string.

null terminator.

char me[6].

char me_big [1000] = "Larry";
= {'L', 'a', 'r', 'r', 'y'}.

char * me = "L"

Closing thoughts about strings.

- The null terminator is used to mark the end of the string.
-

```
char s[1000] = "something";
for (int i=0; s[i] != 0; i++) {
        do_something_with(s[i]);
}
```

Canonical loop over contents of string.

- To compare strings, use strcmp.
- To find length of string use strlen.
- Both declared in <string.h>.