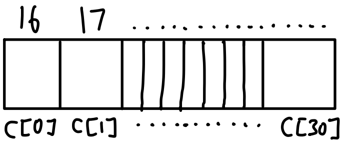
Data Structures

Array

- homogeneous collection of data (string etc.)
- occupies contiguous memory locations
- element type, array name, size

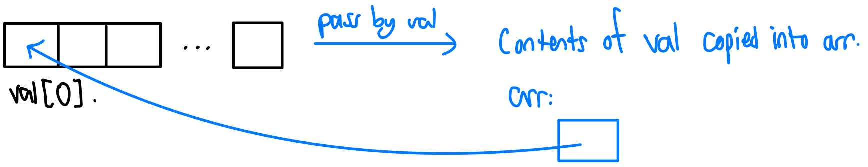


- access through indexing
- insert values to array (in function)
  - ⇒ &arr[i] as argument
- int C[5] = {0};
  - ⇒ initialise all other elements to 0.
- print out arr as arr[0] ⇒ address is identical
- array names are fixed (constant) pointers
  - ⇒ no way to change
  - ⇒ copy all contents of source to destination
    - ⇒ do a for loop or memcpy().
- passing arrays as parameters to functions.

```
int sumArray (int arr[], int);
```

arr[ ] ← val[ ] into — synthetic sugar – i.e. easier to understand. — no need length.

val[ ] into → pointer to the start of array



Contents of val copied into arr.

Function prototype: can leave out argument name & array size.
(ignored)

⇒ need to add in size parameter for array.

alt. syntax for array: sumArray (int*, int);

Since arr (name) is a pointer to first element of the array.

no need to pass address of an array name to the function. → can always modify array contents

Strings

- array of char with null '\0' (ascii val = 0) at the end
- make use of string functions < string.h >
- char fruit_name[ ] = "apple" → will auto-terminate with null.
  - array of 6 chars
  - no need length
- "C" vs 'C'
  - "C" ⇓ length = 2 w/ null
  - 'C' length = 1
- read string: fgets (str, size, stdin) read until size-1 char or newline
  scanf ("%s", str) read until white space

print string: puts(str) terminate with new line (incl.)
printf ("%s\n", str) print till null termination.

strlen ⇒ length of string w/o null.
↳ eliminate null with len-1.

< ctype.h > ⇒ toupper()

strcmp ⇒ compares the ASCII value of 2 strings.
strncmp ⇒ compare until nth character.

strcpy (s1, s2)
- copy from s2 to s1
- up to and incl. null terminator.

Not safe, unable to stop without null terminator.

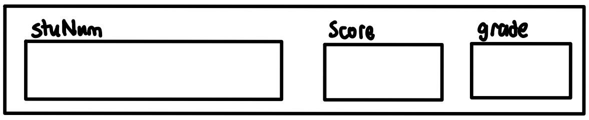
strncpy (s1, s2, n)
- copy at most n character
- n should be one less than s2.length.

%s and string functions will only work on true strings that are null terminated.

Structures

- Allow grouping of heterogeneous members of different types.

- eg.



- a group can be a member of another group
  ⇓
  structure type
  eg.

```
typedef struct {
        int length, width, height;
} box_t;
  name
```

- a type is not a variable - no memory is allocated to a type.
- initialise with {l, w, h}

- use dot operator to access variables eg. box.length.
- read into structure members → & result1.stuNum
- If use structure variable names
  ⇒ unlike array, can do assignments

- Pointers & Structures
  - When pass structure as argument
    ⇒ pass by value (i.e. copied)

- Structure & Array
  - Can have a combination of them.
  - Pass addr. of structure to functions
    - use pointer to structure
      - need to deref ptr.

note: (*player_ptr). age

- else we will deref. player_ptr.age

Synthetic sugar "→"

(*player_ptr).name

|||

player_ptr → name