COMP1511

Static arrays

Week 3 Lecture 2

functions/procedures recap

– Reusable blocks of code
– Callable multiple times
– variables within a function are scoped to that function
– Reusable blocks of code that either do something, or calculate/return something
– Any variables declared within the function (including parameters) are destroyed when the function ends

PI function

Would be annoying to write this every time we need to calculate!

```
double pi() {
  double sum = 0.0;
  for (int i = 0; i < 1000; i++) {
    sum += (-1.0) * pow(1.0 / 2.0, i) / (i + 1);
  }
  return 4.0 * sum;
}
```

Forward declaration

```
int main(void) {
    double calculated_pi = pi();
}

double pi() {
  double sum = 0.0;
  for (int i = 0; i < 1000; i++) {
    sum += (-1.0) * pow(1.0 / 2.0, i) / (i + 1);
  }
  return 4.0 * sum;
```

```
}
```

^ problem! `main` doesn't know that pi exists yet!

Forward declaration

```
double pi();


int main(void) {
    double calculated_pi = pi();
}


double pi() {
  double sum = 0.0;
  for (int i = 0; i < 1000; i++) {
    sum += (-1.0) * pow(1.0 / 2.0, i) / (i + 1);
  }
  return 4.0 * sum;
}
```

^ Solved! We forward declared pi!

Quick functions recap demo

Arrays

So far, we can store a single item in each variable

What if you wanted to store many values?

Number of ice creams eaten

```
int day_1 = 2;
int day_2 = 3;
int day_3 = 3;
int day_4 = 5;
int day_5 = 7;
int day_6 = 1;
int day_7 = 3;
// Any day with 3 or more scoops is too much!
if (day_1 >= 3){
```

```
    printf("Too much ice cream\n");
}
if (day_2 >= 3) {...
```

Seem repetitive?

Remember, if anything seems repetitive, there is often a cleaner solution

–

Many variables would clutter the program

Think if we need to store 1,000 students in COMP1511 course, do we want a variable for each student?

–

Many variables would not always be efficient

The memory allocations would be scattered around the RAM

Data structures

– Are common structures (not structs) used to store multiples of data

  – Usually (especially in COMP1511) of the same data type

– Can scale, easily storing a handful, up to thousands, or more elements of data!

Data structures in COMP1511

We will look primarily at two data structures:

– arrays (today)

–

linked lists (future)

These are very, very powerful data structures you will use forever

Arrays

– A collection of data, all of the same type. (homogonous)

– We have a single identifier for the entire array

– It is a random access data structure, meaning we can access any element in the array at any time

Arrays

– We can ready or modify individual elements

– It is a ***contiguous*** data structure

contigu-what?

Let's visualise arrays

See whiteboard for visualisation

– Our C file is stored on the hard drive

– Our Compiler compiles the code into another file that the computer can read
– When we execute code, the CPU will actually process the instructions and perform basic arithmetic, but the RAM will keep track of all the data needed in those instructions and operations, such as our variables.
– Reading and writing to variables will change the numbers in RAM
–
Memory is divided into the stack and the heap
The stack is an ordered stack and the heap is a random free for all - insert something where you can find space for it.

Static arrays have a set size

(which you specify)

index:	0	1	2	3	4
values:					

int array

index:	0	1	2	3	4
values:					

– This int array will store 4 integers
– 32bit * 4 elements = 128 bits of memory used

The array declaration syntax

```
int ice_cream_per_day[7];
```

index:	0	1	2	3	4	5	6
values:							

Asks the operating system for a piece of memory big enough to store 7 integers. Note that there are 7 elements, but index only goes up to 6!

Declare + initialise

```
int ice_cream_per_day[7] = {3, 2, 1, 2, 1, 3, 5};
```

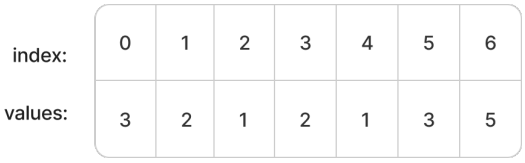
^ Note you can only do this when you declare, not later!

```
int ice_cream_per_day[7] = {};
```

^ Will initialise all elements to 0

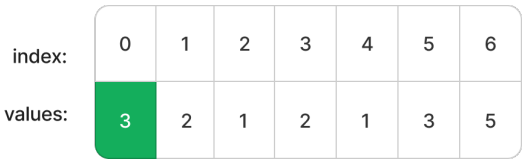
```
int ice_cream_per_day[7] = {3, 2, 1, 2, 1, 3, 5};
```

Creates:



Accessing elements

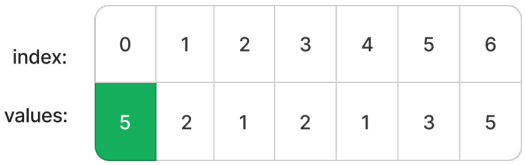
```
int first_day_ice_creams = ice_cream_per_day[0];
```



Will retrieve the int 3

Writing elements

```
ice_cream_per_day[0] = 5;
```



Will write to index 0 of ice_cream_per_day

arrays 🤝 loops

The power of arrays

```
int ice_cream_per_day[7] = {3, 2, 1, 2, 1, 3, 5};

// read each element
ice_cream_per_day[0];
ice_cream_per_day[1];
ice_cream_per_day[2];
ice_cream_per_day[3];
ice_cream_per_day[4];
ice_cream_per_day[5];
```

```
ice_cream_per_day[6];
```

^ Does this look repetitive?

If only we had a way to count :(

Bad

```
int ice_cream_per_day[7] = {3, 2, 1, 2, 1, 3, 5};

// read each element
printf("%d\n", ice_cream_per_day[0]);
printf("%d\n", ice_cream_per_day[1]);
printf("%d\n", ice_cream_per_day[2]);
printf("%d\n", ice_cream_per_day[3]);
printf("%d\n", ice_cream_per_day[4]);
printf("%d\n", ice_cream_per_day[5]);
printf("%d\n", ice_cream_per_day[6]);
```

Good

```
int ice_cream_per_day[7] = {3, 2, 1, 2, 1, 3, 5};

int i = 0;
while (i < 7) {
    printf("%d\n", ice_cream_per_day[i]);
    i++; // i = i + 1;
}
```

Demo

Feedback

https://forms.office.com/r/Ze4admEWnR


COMP1511 23T3 Lecture Feedback