Declaring and initialising

- We already talked about how to declare an array.
- For example:

```
int age[8];
```

declares an array of 10 `int`s called `age`.

- We can have arrays of any data type. For example:

```
double numbers[5];
char   sentence[30];
```

declares an array of 5 `double`s called `number`, and an array of 30 `char`s called `sentence`.

- We can initialise these arrays when we declare them, just as we can for other kinds of variable:

```
int age[8] = {5, 10, 15, 20, 25, 30, 35, 40}
```

- Initialising like means that:

```
cout << age[0] << endl;
cout << age[6] << endl;
```

will produce:

```
5
35
```

since, as we mentioned before, the first element in the array has subscript 0.

ARRAYS

Today

- We have already touched on arrays, back in lecture III.1.
- Today we will begin to look at arrays in more detail.
- These notes will make most sense if you go back and read the notes for lecture III.1 first.
- Many of the examples in the notes are contained in the program `more-arrays.cpp` which can be downloaded from the class web page.

Handling arrays

- Arrays are commonly handled using `for` loops.
- Thus, to print out the array `age`, we might use:

```
for(counter = 0; counter < 8; counter++)
    {
      cout << age[counter] << "  ";
    }
```

- This will take each element of `age` in turn and send it to `cout`.
- Note that the maximum value of the subscript `counter` is 7.
- This is because although `age` is 8 elements long, the subscript of the first element is 0.
- *In computer science we start counting at 0.*

- In C++ it is very important to be careful with the maximum value of the subscript of an array.
- If you *overflow* an array, for example by doing:

```
age[10] = 30;
```

this will not generate an error.

- However, it may cause your program to crash in an unexpected way.

- If instead we initialised with:

```
int age[10] = {5, 10, 15, 20, 25, 30}
```

then

```
cout << age[0] << endl;
cout << age[6] << endl;
```

would produce:

```
5
0
```

since any values we do not explicitly assign in the initialisation will be set to 0.

- Declaring:

```
double number[5]    = {5.0, 10.1, 20.2, 30.3};
```

would set `number[4]` to 0.

- Declaring:

```
char   sentence[30] = {'H', 'e', 'l', 'l', 'o'}
```

would set all elements of `sentence` after the `'o'` to `' '`.

- Often, when handling arrays, we want to use the same subscript to access two or more arrays.
- For example:

```
for(counter = 0; counter < 5; counter++)
  {
    number[counter] = 2 * age[counter];
  }
```

- This replaces each element of `number` with double the corresponding element of `age`

Casting

- Note that it is safe to assign the elements of `age` to `number` because `number` contains `doubles`, and we can use a `double` to hold an integer.
- However, if we do:

```
for(counter = 0; counter < 5; counter++)
  {
    age[counter] = number[counter];
  }
```

We will get unpredicatble results because there is not enough room in a `int` to hold all the information in a `double`.

- What we can do is to deliberately exclude the decimal part of `number`.
- We do this using an operation called *casting*.

- In this code:

```
for(counter = 0; counter < 5; counter++)
  {
    age[counter] = (int)number[counter];
  }
```

We cast a `double` into an `int`, *losing* information (the decimal part).

- We can also cast to *gain* information.

- For example:

```
int sum    = 203;
int count  = 20;
double average;

average = sum/count;
cout << average;
```

will output 10, since the division is integer division, and so will generate an integer answer.

- Altering the division to

```
average = ((double)sum)/count;
```

will temporarily make `sum` a double, and so the division will be a double divided by an integer, which will give a decimal answer that can be assigned to `average`.

Manipulating subscripts

- The subscript that we use to identify elements of an array is also just an integer.
- So we can use arithmetic expressions as subscripts, *so long as they eveluate to integers*.
- For example

```
cout << age[2+1];
cout << age[counter - 2];
cout << age[age[0]];
```


Using arrays

- The homework will explore the use of arrays in a biomedical context.
- Lots of recent biomedical research has concentrated on analysing genetic information — information encoded in DNA.
- You can think of DNA as being long sequences of letters drawn from an alphabet of four letters, C, A, T and G.
- Clearly we can represent such sequences as arrays of characters:

```
char dna[7] = {'a', 't', 'a', 't', 'a', 'g', 'c'}
```


Modifying elements in an array

- Of course, since each element of `age` is an integer, we can do to each element, exactly what we can do to an integer.
- All we have to do is remember how to address each element of the array, using a subscript.
- For example:

```
for(counter = 0; counter < 8; counter++)
  {
    age[counter]++;
    age[counter] = (age[counter] * 2)/3;
  }
```

- What does this do to each element of the array `age`?

Functions and array elements

- Since elements of `age` are integers, we can call functions on them
- If we have the function

```
int timesTwo(int number)
{
  return 2 * number;
}
```

we can call this on the third member of `age` like so:

```
timesTwo(age[2]);
```

Summary

- This lecture has looked in more detail at arrays.
- We examined the initialization of arrays.
- We looked at handling arrays using `for` loops, and by playing with subsscript values.
- We looked at different things one can do with array elements.
- Along the way we also looked at casting.
- We finished by sketching one use for arrays in a biomedical context.

- What we typically want to do with DNA sequences is to search for patterns in them, and C++ gives us the tools to do this.
- For example:

```
for{counter = 0; counter < 4; counter++)
  {
    if(dna[counter]     == 't' &&
       dna[counter + 1] == 'a' &&
       dna[counter + 2] == 'g')
       {
          cout << "We found tag";
       }
  }
```

will search `dna` for the sequence `tag`.

- To do more complex searches, we need better ways of handling sequences of characters, and we will start to look at those ways in the next lecture.