Programming Principles

Unit 8

Arrays

Arrays

What are they?

Arrays

A way of organising data into numbered Lists

Why do we need arrays?

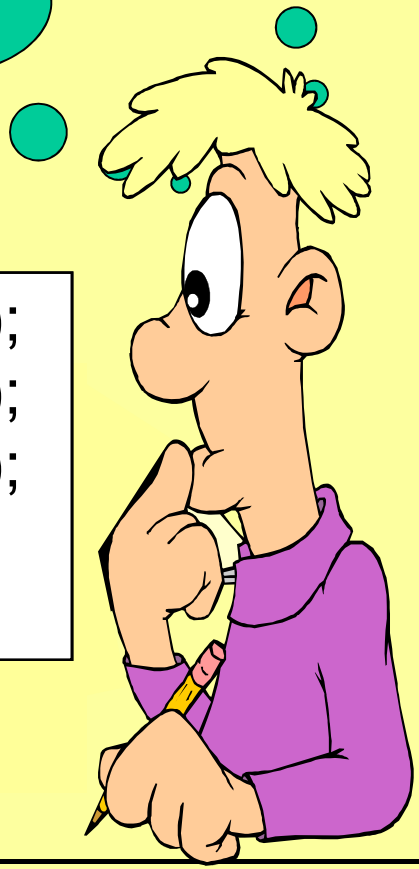
Dumb Programming Method

We want to store the ages of a class of 30 students

```
int age1 ;
int age2 ;
int age3 ;
int age4 ;
etc.
```

What if there are 300 students?

There must be a better way!



```
age1 = dialog ( "Enter age of student 1" );
age2 = dialog ( "Enter age of student 2" );
age3 = dialog ( "Enter age of student 3" );

etc.
```

Array Method

Define the Age array

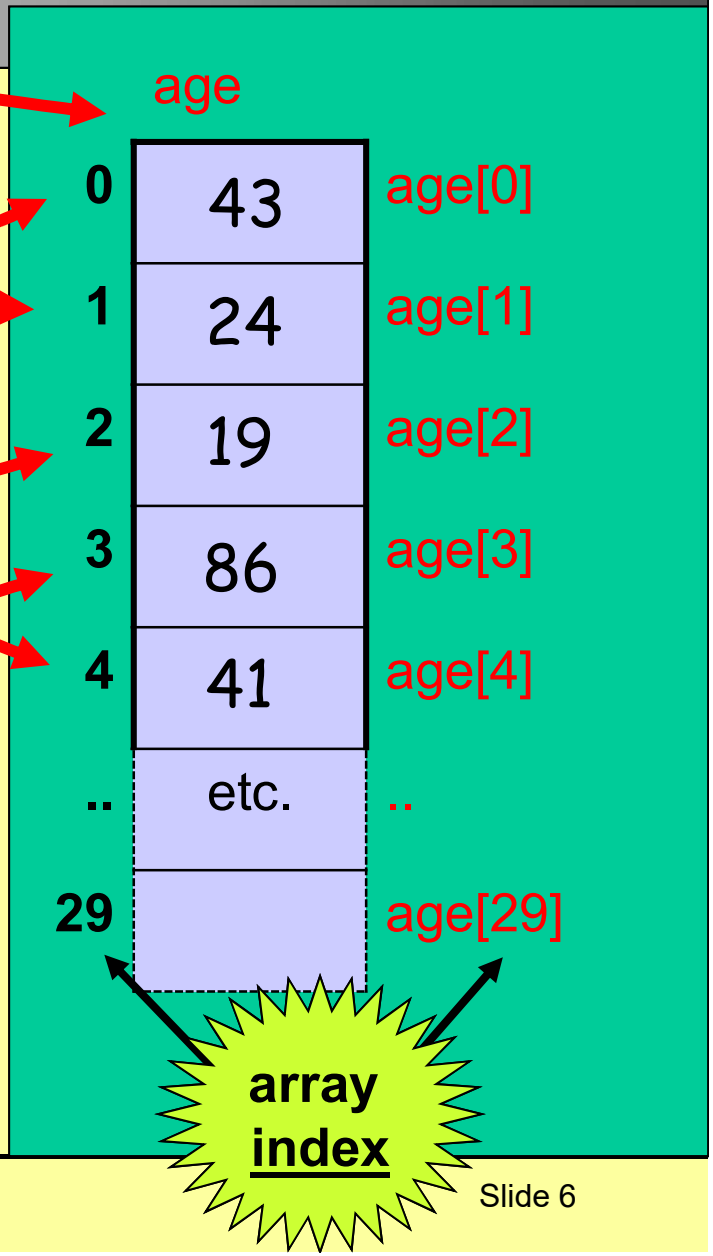
int age[30] ;

Input to the array

```
age [ 1 ]  = 24 ;
age [ 4 ]  = 41 ;
age [ 2 ] = dialog(" Your age?") ;
age [ 0 ] = age[ 2 ] + age [ 1 ]  ;
age [ 3 ] = age[ 0 ] * 2  ;
```

Output from the array

```
message( "The first age is " + age [ 0 ]  ) ;
message( "The last age is " + age [ 29 ] ) ;
```



Important Note

Defining an array of 30 integers

```
int   age[30];
```

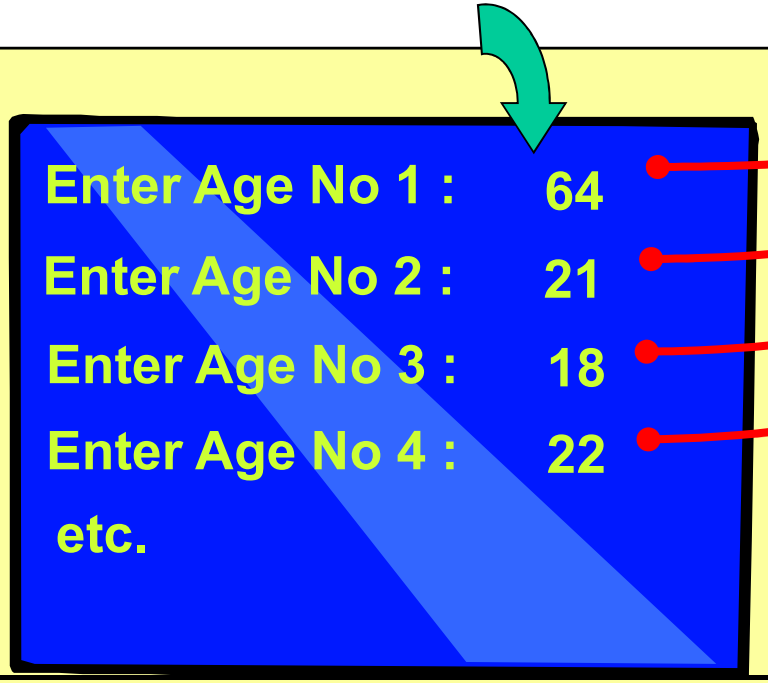
In this case the array is called age and it has 30 elements (from 0 up to 29).

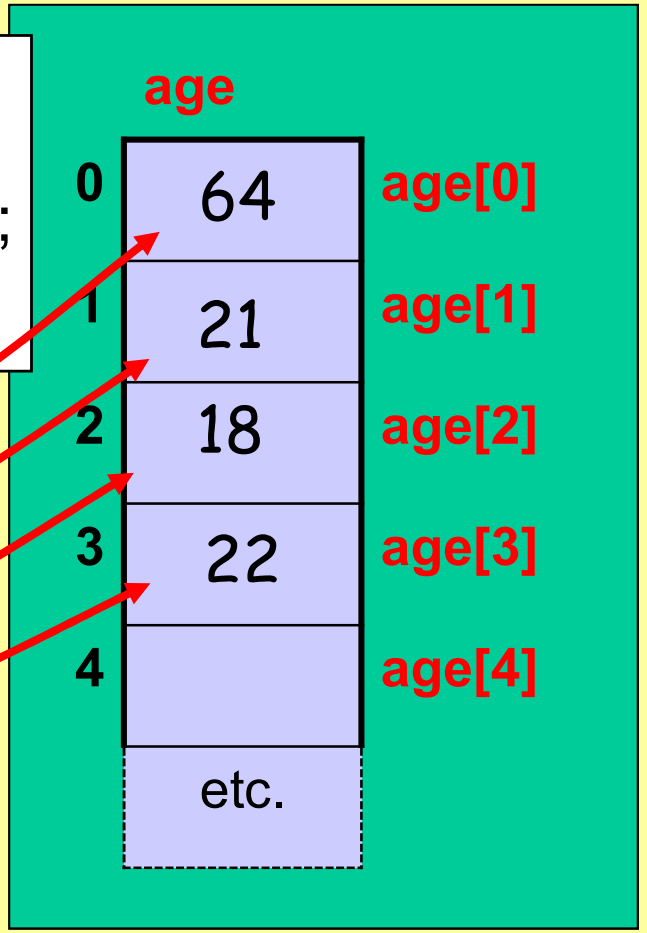
Position 30 in the array does NOT exist

Filling the age Array

A for-loop can be used to enter data into the whole age array

```
for (i = 0 ; i < 30 ; i++)
{
   age[i] = dialog( "Enter Age No : " + (i+1) );
}
```





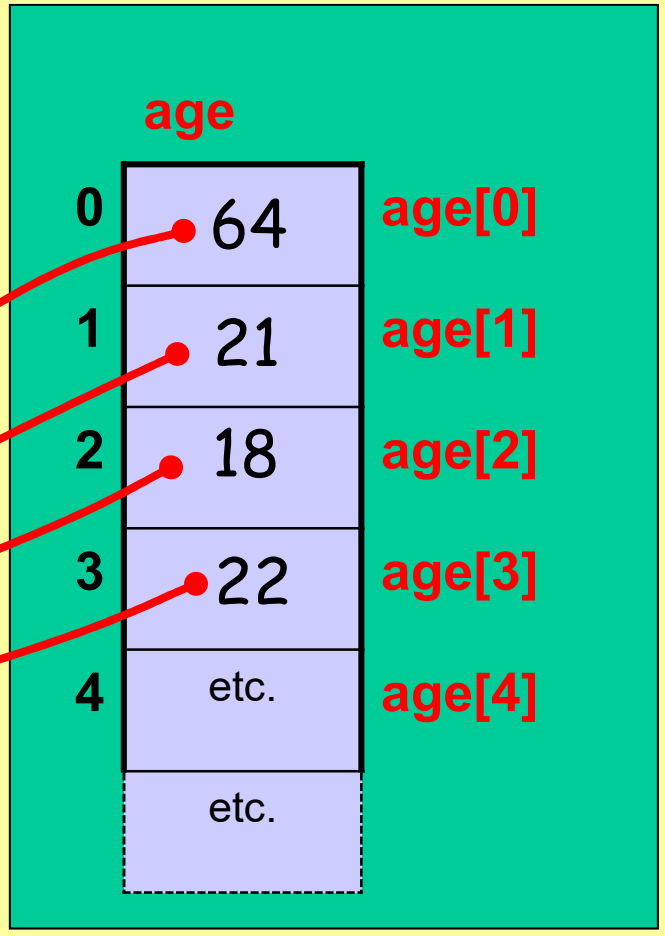
Output of the whole age Array

A for-loop can be used to print the whole age array contents

```
for (i = 0 ; i < 30 ; i++)
 {
    message( "Age No " + (i+1)
                    + " is " + age[i] );
 }
```

```
Age No 1 is   64
Age No 2 is   21
Age No 3 is   18
Age No 4 is   22

etc.
```

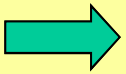


Other Types of Array

We can produce arrays using any of the usual data types: e.g.

```
int age[30];            // defines an array of 30 integers
float wage[20];         //  an array of 20 float numbers
object items[100];      //  an array of 100 objects
string names[20];       //  an array of 20 strings
```

Each element of an array can be accessed using the array index (the integer variable i in the previous slides)



```
age [ i ]
wage [ i ]
items [ i ]
names [ i ]
```

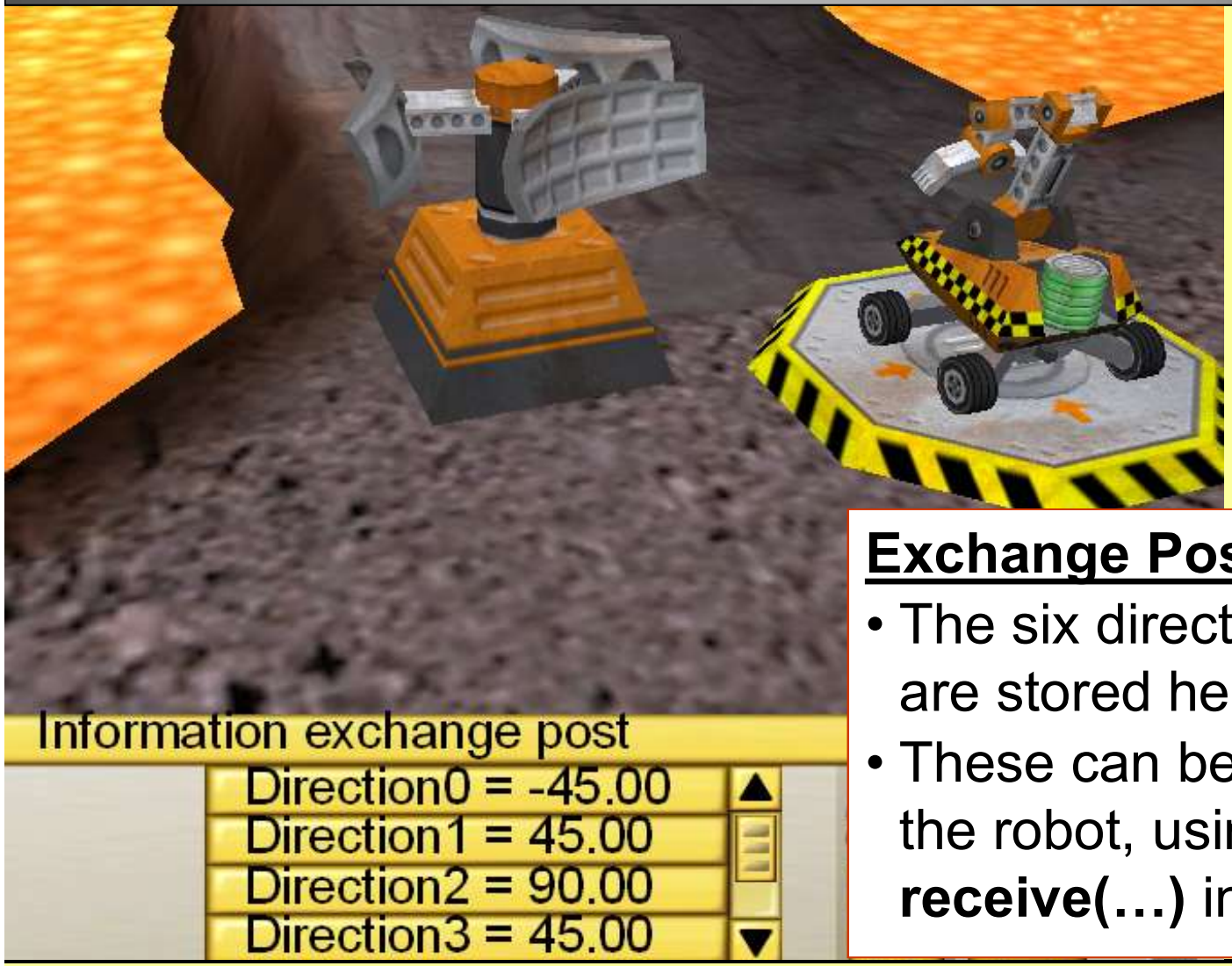
A Ceebot example

Exchange Posts

The Robot's Path

- There are six 20 metre sections to the path
- After each section the robot must **turn** .. but by how much?

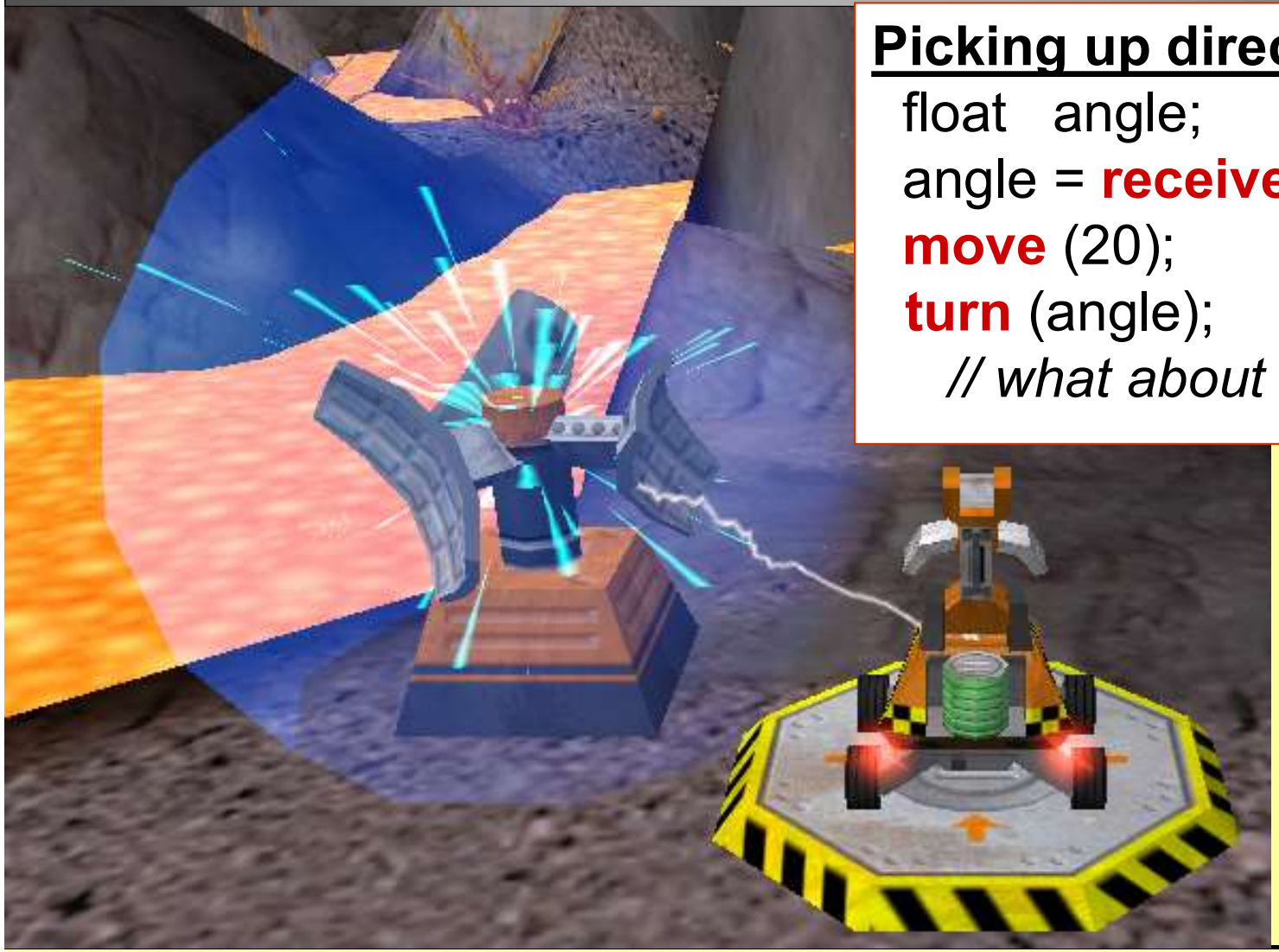
Directions Stored



Exchange Post

- The six directions (angles) are stored here
- These can be picked up by the robot, using the **receive(…)** instruction

The receive(..) instruction



Picking up directions?

```
float   angle;
angle = receive("Direction0");
move (20);
turn (angle);
   // what about other angles?
```

Program using an array

```
extern void object::arrayexample()
{
    float  dir[6];    // define array called dir to hold 6 directions
    // receive and store all directions from exchange post
    for ( int i = 0;  i < 6;  i ++ )
    {
        dir [ i ] = receive ( "Direction" + i ) ;
    }
    // now start moving along the path
    for ( int i = 0;  i < 6;  i ++ )
    {
        move ( 20 ) ;
        turn ( dir [ i ] ) ;
    }
}
```

Loop 6 times

turn using angles now stored in the array

Arrays and functions

(passing whole arrays into functions)

```
extern void object::ArraysAndFunctions()
{
    float  dir [6];    dir[ 0 ] = 0;
    getData(  dir );
    useData( dir );
}
```

must initialise the array in Ceebot

```
void object::getData( float[ ]  data)
{
    for (int i=0; i<6; i++)
    {
        data[i] = receive( "Direction" + i );
    }
}
```

The array **dir** is passed as a parameter to the array **data**

```
void object::useData(float[ ]  data)
{
    for (int i=0; i<6; i++)
    {
        move (20);
        turn ( data[i] );
    }
}
```

Look-up Tables

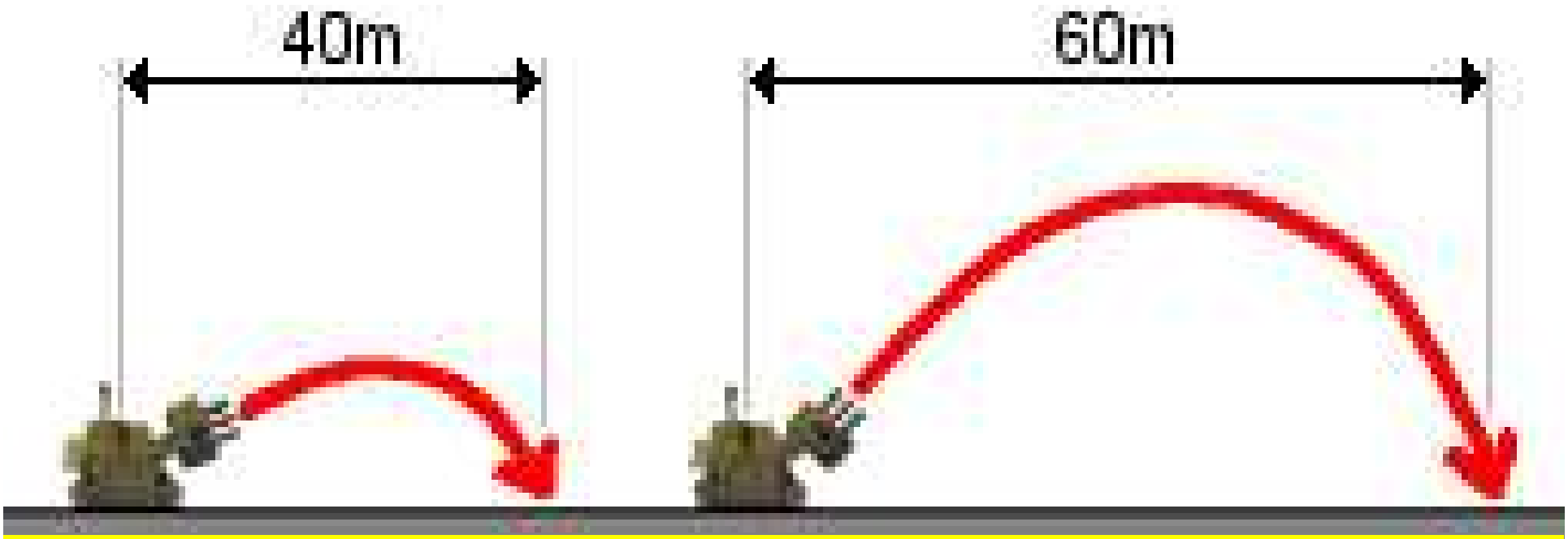
(using arrays to look up information)

Ballistic Fire using a PhazerShooter

The PhazerShooter can fire at large angles (up to 45 degrees)

How can we find the correct angle?

```
aim(20);   aim(45);
```



- To reach a distance of 40m, aim at an angle of 20 degrees
  To reach 60m, we should use an angle of 45 degrees
  What about all the other angles?

Look-up Table

float angle[64];

angle[0] = 0.0; angle[1] = 0.45; angle[2] = 0.90; angle[3] = 1.35;
angle[4] = 1.80; angle[5] = 2.25; angle[6] = 2.70; angle[7] = 3.15;
angle[8] = 3.61; angle[9] = 4.06; angle[10] = 4.52; angle[11] = 4.97;
angle[12] = 5.43; angle[13] = 5.89; angle[14] = 6.35; angle[15] = 6.81;
angle[16] = 7.27; angle[17] = 7.74; angle[18] = 8.21; angle[19] = 8.68;
angle[20] = 9.15; angle[21] = 9.62; angle[22] = 10.10; angle[23] = 10.58;
angle[24] = 11.06; angle[25] = 11.55; angle[26] = 12.04; angle[27] = 12.54;
angle[28] = 13.04; angle[29] = 13.54; angle[30] = 14.05; angle[31] = 14.56;
angle[32] = 15.08; angle[33] = 15.60; angle[34] = 16.13; angle[35] = 16.66;
angle[36] = 17.20; angle[37] = 17.75; angle[38] = 18.31; angle[39] = 18.87;
angle[40] = 19.45; angle[41] = 20.03; angle[42] = 20.62; angle[43] = 21.22;
angle[44] = 21.84; angle[45] = 22.47; angle[46] = 23.11; angle[47] = 23.77;
angle[48] = 24.44; angle[49] = 25.14; angle[50] = 25.85; angle[51] = 26.59;
angle[52] = 27.35; angle[53] = 28.15; angle[54] = 28.97; angle[55] = 29.84;

- **This is an array that is able to hold 64 numbers**
  The correct angles for all distances up to 64 metres is stored
  e.g. If the distance is 26 metres, the angle should be 12.04

Part of the Phazer Program

Define variables

```
object alien;
float  dist, fireangle;
float  angle[64];  // set up for you!
```

1. **Detect a spider**

```
alien = radar(AlienSpider);
```

2. **Turn towards spider**

```
turn( direction( alien.position ) );
```

3. **Work out distance to spider**

```
dist = distance( this.position,  alien.position    );
```

4. **Look up the firing angle in array**

```
fireangle = angle[ dist ];
```

5. **Aim gun using this angle**

```
aim( fireangle );
```

All of this will need to be repeated in a loop

6. **Destroy Spider**

```
fire(1);
wait(1);    // allow time for shell travel
```


What I learned today
1. Arrays as lists
2. Arrays as data structures
3. Defining arrays
4. Array input and output
5. Array index
6. Exchange posts and receive() instruction
7. Arrays as function parameters
8. Arrays as Look-up tables
THE END

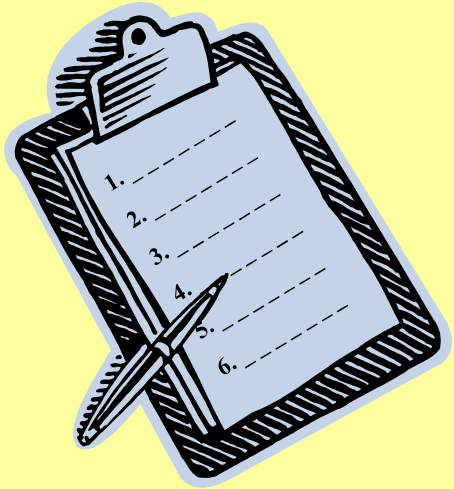
Arrays are Data Structures

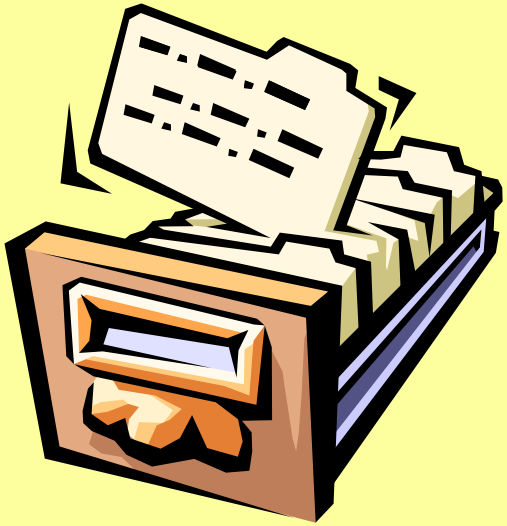
We can combine simple data types into more complex structures

Array
a numbered list of similar data types

Class
a single package to hold data and functions (methods) for an object

File
long-term storage for data





Arrays
(numbered lists)

Array Names

The **name** of an array holds the **address** of the start of the array

e.g. int age[30];

age holds the start address for the array

To pass an array to a function, we only need to pass its **name**

Any changes made inside the function will automatically change the calling array

Two-Dimensional Arrays
(tables)

Arrays with 2 Dimensions

2-dimensional arrays look like tables with rows and columns

Define the Array

```
int  marks [3][5] ;
```

marks array

	0	1	2	3	4
0		24			
1	43				41
2				19	

Input to the array

```
marks [0] [1]  = 24 ;
marks [1] [4]  = 41 ;
marks [2][3] = dialog("Enter mark");
marks [1][0] = marks [0][1] + marks [2][3] ;
```

Output from the array

```
message( "The  mark for student 5 of class 2 is " + marks [1][4] ) ;
```

Filling the 2-D marks Array

To fill the whole marks array, we need to use 2 for-loops

```
for ( i = 0; i < 3; i++)          // 3 classes
  for ( j = 0; j < 5; j++)        // 5 students per class
   {
       marks[i][j] = dialog("Enter mark for class " + i
                                + " student " + j ) ;
   }
```

marks array

	0	1	2	3	4
0	64	78	56	77	83
1	46				
2					

```
Enter mark for class 0 student 0   64
Enter mark for class 0 student 1   78
Enter mark for class 0 student 2   56
Enter mark for class 0 student 3   77
Enter mark for class 0 student 4   83
Enter mark for class 1 student 0   46
  etc.
```

the code uses nested for-loops