# CO452 Programming Concepts

Week 6 - Parameters

## Recap of the last week

Last week we looked at:

- User-defined functions
- Program design
- Scope of variables

## Aims and Objectives

**Aim:** to expand on functions and explore passing data between them to aid efficiency and data control

**Learning outcomes:**

- To know how to pass and return data between functions
- Create efficient solutions to problems in Ceebot

## This week

Continuing with functions:

- Passing parameters by-value with our functions
- Returning values
- Formal and actual parameters

## User-defined functions

***Designing your own functions***

## Our functions

```
extern void object::task18_3(){
   functionName();   //call function


}
void object::functionName(){
    message("Hello World");


}
```

# Passing parameters by-value

*Customising functions*

## An example we're familiar with

*function name*

*Parameter*

```
move(20);
```

## Passing strings

```
extern void object::task18_3(){
    string message = “Hello World”;
    outputString(message);     //call
}

void object::outputString(string text){
     message(text);
}
```

## Passing integers

```
extern void object::task18_3(){
    int num = 10;
    doubleNum(num);    //call
}

void object::doubleNum(int num){
     message(num * 2);
}
```

## Reminder about local variables

## How many variables?

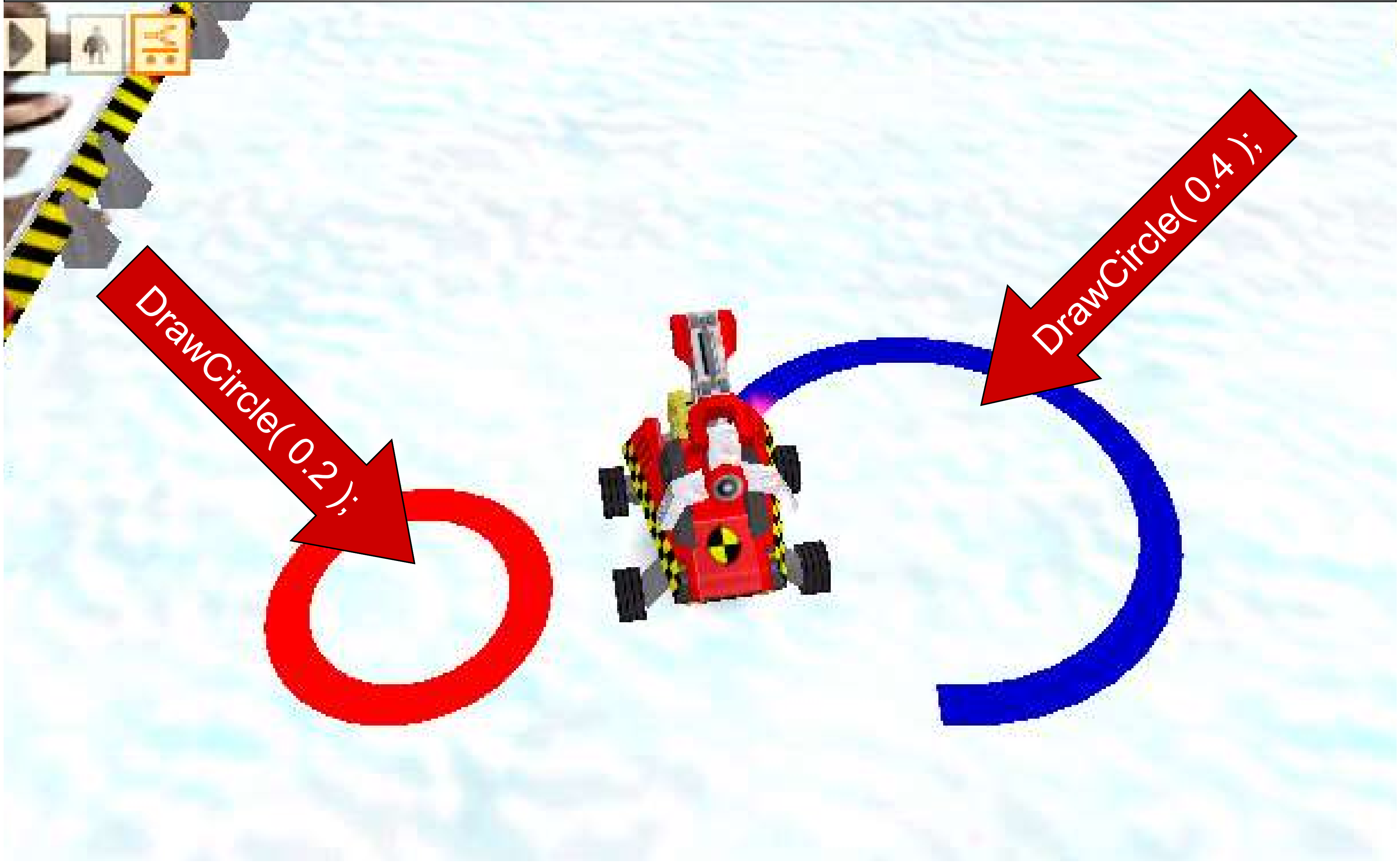
```
extern void object::task18_3(){
     int num = 10;
     doubleNum(num);     //call
}

void object::doubleNum(int num){
      message(num * 2);
}
```

## Using parameters in our own functions

*Making them more flexible*

# A better DrawCircle() function



# Using new drawCircle()

```
extern void object:: DrawUsingParameters()
{
        red();
        drawCircle(0.2);
        move(5);
        blue();
        drawCircle(0.4);
}
```

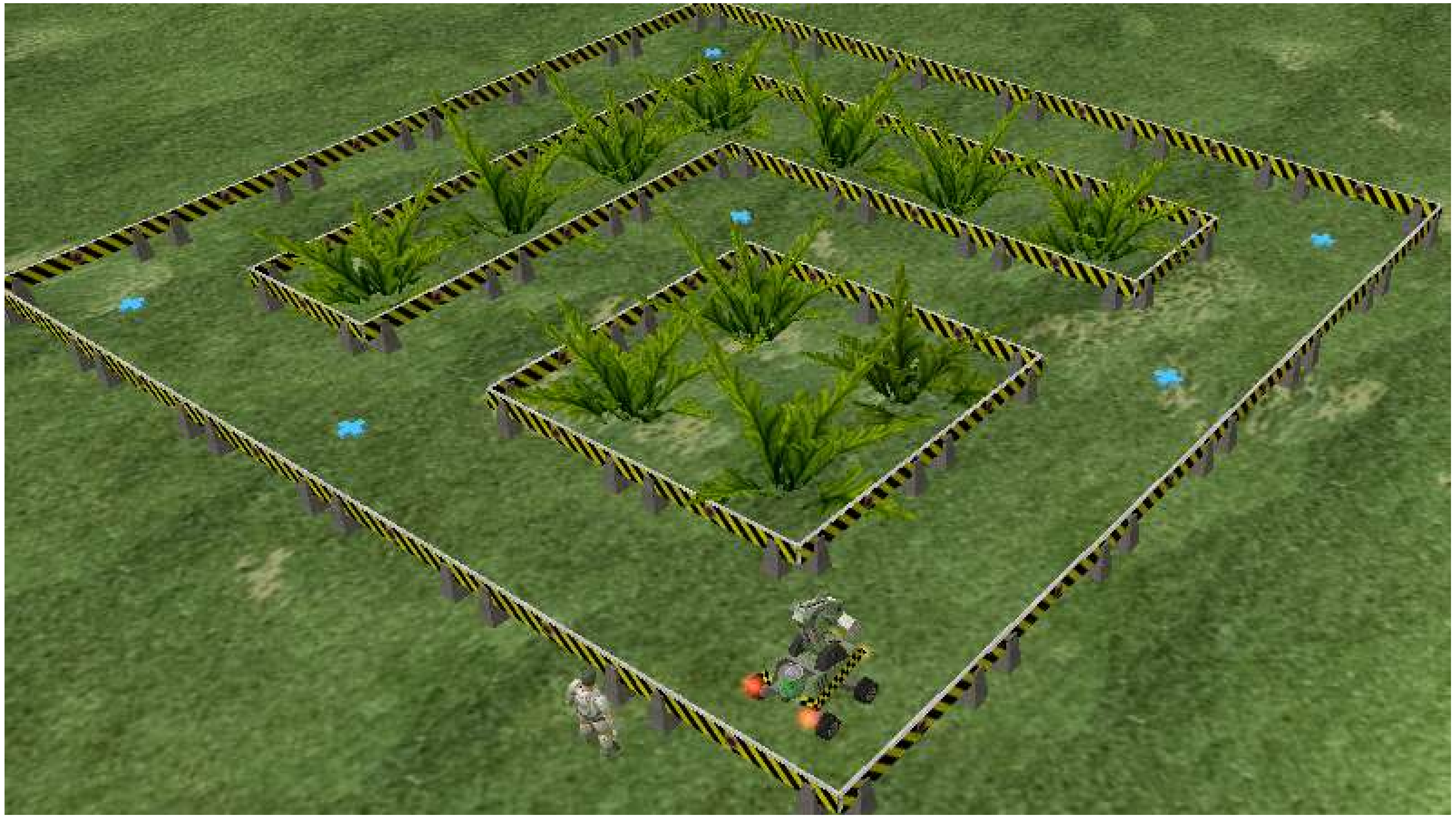
This main program calls drawCircle() twice with 2 different parameters

```
void object:: drawCircle (float  step)
{
        pendown();
        for (int i = 0; i < 36; i++)
        {
                move(step) ;
                turn(10) ;
        }
        penup();
}
```

step takes the value 0.2 then 0.4

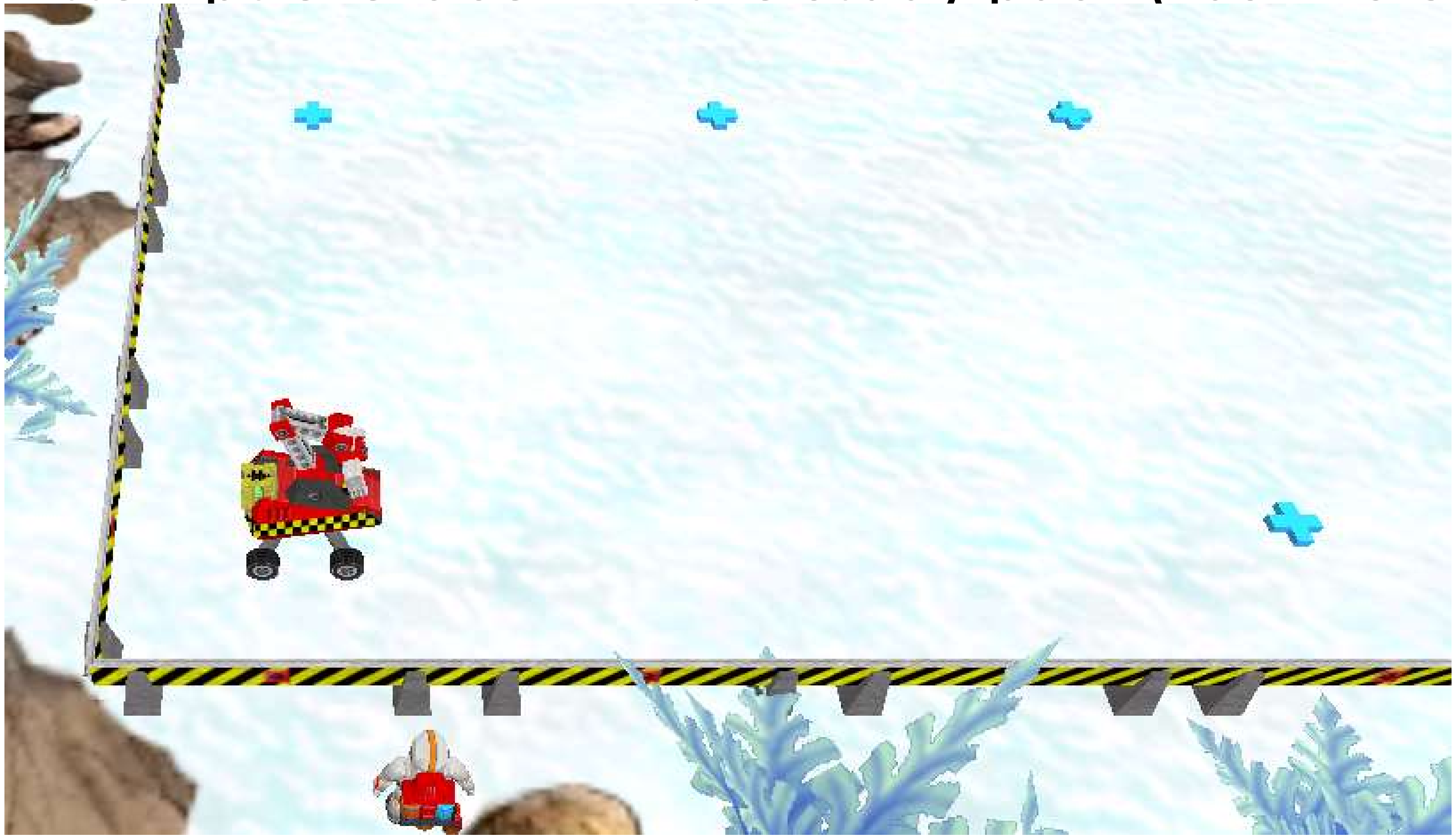
# Activity

Attempt exercise 1 in the study pack (Task 20.1)



## Activity

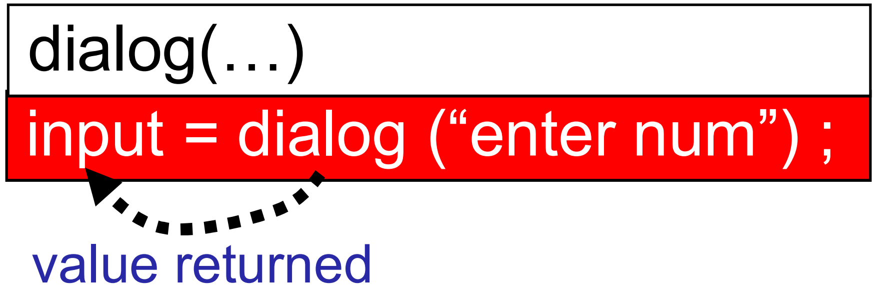
Attempt exercise 2 in the study pack (Task 20.3)



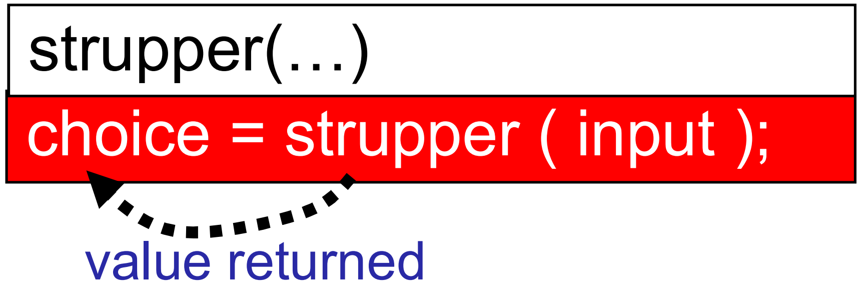
## Returning values

***Passing control back***

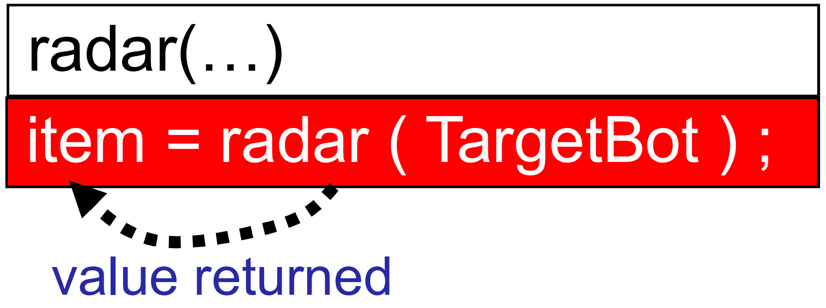
# Remember these?



```
dialog(…)
input = dialog (“enter num”) ;
```



```
strupper(…)
choice = strupper ( input );
```



```
radar(…)
item = radar ( TargetBot ) ;
```

## Returning integers

```
extern void object::task18_3(){
    int num1 = 10, num2 = 20, total = 0;
    total = addNum(num1, num2);
    message(total);
}
int object::addNum(int num1, int num2){
     int total = num1 + num2;
     return total;
}
```

## Returning data

The **return type** that the function is declared with has to **match** the **type** of data that is being returned.

```
extern void object::mainProgram2()
{     float num1, num2, avg;
      num1 = getNum ("Enter first number");
      num2 = getNum ("Enter second number");
      avg = calcAverage(num1, num2 );
      message (num1 + " and " + num2 + " average is " + avg);
}
```

```
float object::calcAverage(float a, float b )
{
      float result = (a + b) / 2;
      return result;

}
```

```
float  object::getNum( string prompt )
{
       float  number = strval(dialog( prompt ) ) ;
       return  number;
}
```

Could actually return the calculation without storing in a variable

## Actual and Formal parameters

***What's the difference?***

# Actual & Formal Parameters

```
extern void object:: Actual&FormalParameters()
{
        red();
        drawCircle(0.2);
        move(5);
        blue();
        drawCircle(0.4);
}
```

0.2 and 0.4 are actual parameters

```
void object:: drawCircle (float  step)
{
        pendown();
        for (int i = 0; i < 36; i++)
        {
                move(step) ;
                turn(10) ;
        }
        penup();
}
```

step is the formal parameter

# Activity

Attempt exercise 3 in the study pack (Task 20.7)

## Modulus

*Alternating paths*

## Alternate Colours?

# Defining a setColour(…) function

```
void object:: setColour ( ) int  loopNum  )
 {
        //  set a different colour
        // depending on the parameter passed in
         int  rem;
         rem = loopNum % 2;
          if (rem == 0)
          {
                     red();
          }
          else if (rem == 1)
          {
                     blue();
          }
 }
```

Modulo arithmetic

% divides (by 2) and leaves the remainder

If you divide by 2 the only possible remainders are 0 and 1

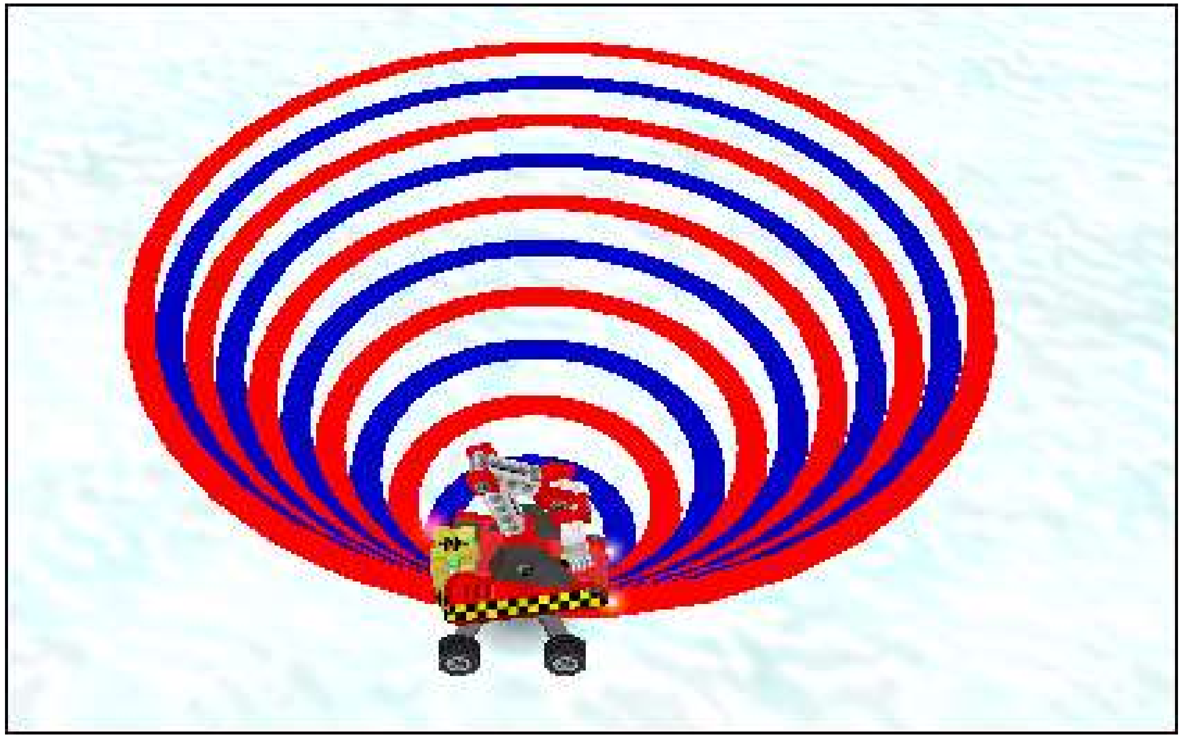
# Using setColour()

```
extern void object::Draw6()
{      red();
       float  size=0.2;
       for (int i=0; i<6; i++)
       {      setColour(   i);                 // pass loop counter i
              drawCircle(  size);              // pass size as a parameter
              size = size + 0.1;               // increase circle size
       }
}
```

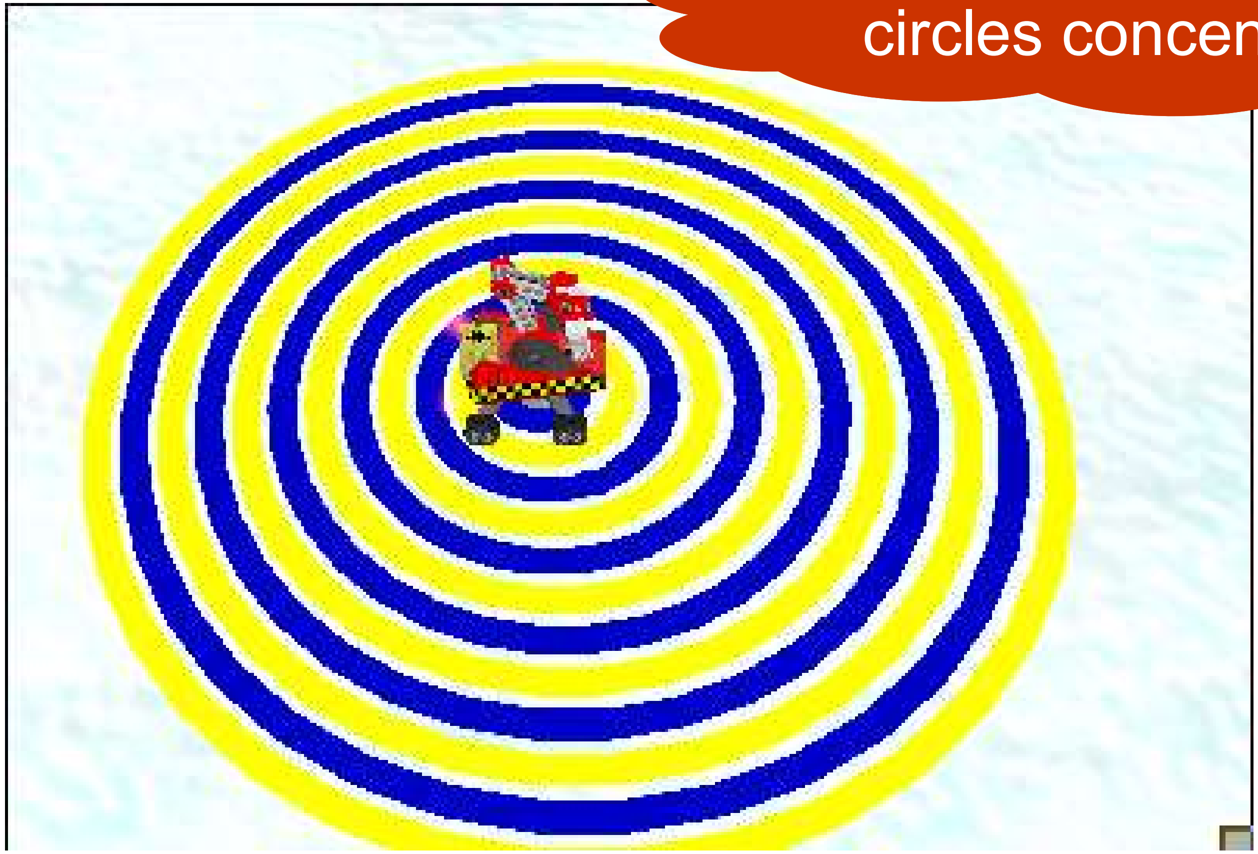
```
void object:: setColour ( ) int  loopNum )
 {
        //  set a different colour
        // depending on the parameter passed in
        int  rem;
        rem = loopNum % 2;
         if (rem == 0)
```

# Activity

Use the space in Task 18.3 to replicate the picture below:



## Challenge



How can we make the circles concentric?

## Quiz

Can a void return type be used when returning a value?

## Quiz

What does passing parameters **by-value** mean?

## Recap

This week we looked at:

- Passing parameters by-value with our functions
- Returning values
- Formal and actual parameters

Extra Reading

## Why use functions?

## Why use functions?

- Large programs can be broken up into smaller sections
- Programs are then easier  to understand
- It is easier to modify programs
- It is easier to locate errors
- division of work among programming teams is easier
- functions can be re-used in other programs
- saves duplicating code (write <u>once</u> .. use <u>many</u> times)
- creates better program structure
- makes programs more:

    readable
    maintainable
    reliable
    and less complex

## Local Variables

- These are declared inside a function

  -- and can only be used in that function

  -- they are not recognised outside the function
- Local variables are created when the function is called

  -- and destroyed when the function finishes
- They help to make functions more independent

  -- so they can be used in other programs without messing them up
- We say that the scope of the variable is the function in which it is declared

## Why use parameters?

## Why use parameters?

- Functions are much more powerful and versatile
- Functions can more easily be re-used in other programs