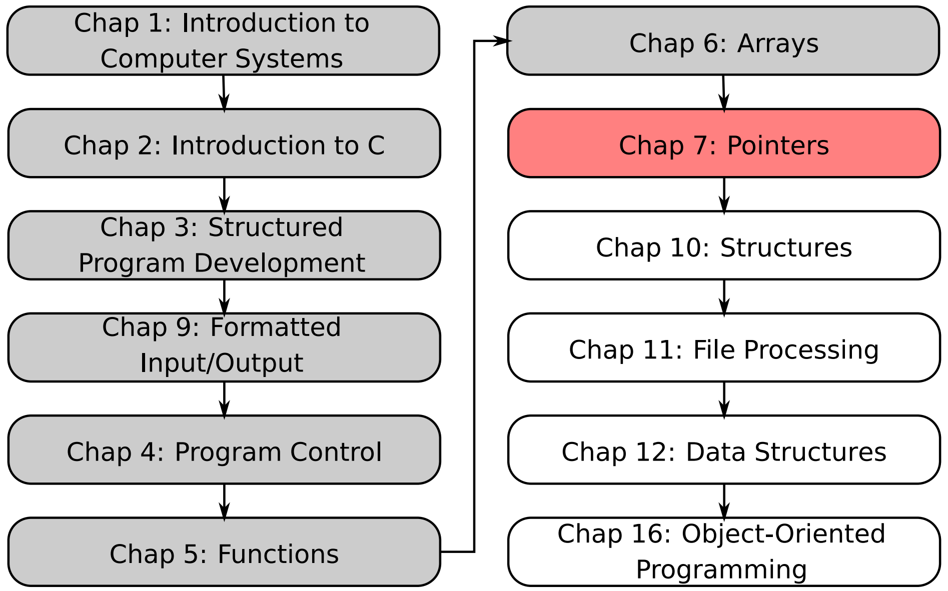
Computer Programming 143 – Lecture 19
Pointers II

Electrical and Electronic Engineering Department
University of Stellenbosch

Corné van Daalen
Thinus Booysen
Willie Smit
Willem Jordaan

Module Overview


Chap 1: Introduction to Computer Systems
Chap 2: Introduction to C
Chap 3: Structured Program Development
Chap 9: Formatted Input/Output
Chap 4: Program Control
Chap 5: Functions
Chap 6: Arrays
Chap 7: Pointers
Chap 10: Structures
Chap 11: File Processing
Chap 12: Data Structures
Chap 16: Object-Oriented Programming

Lecture Overview

7.1-7.3 Review of Pointer Basics

Pointer Basics

- A pointer is a variable that stores a memory address
  - The pointer "points to" the variable at its stored memory address
- Declaration of a pointer:

```
int *myPtr;
```

  - Declares variable `myPtr` as a pointer to an integer
- Address operator (&):

```
myPtr = &myInt;
```

  - Stores the address of `myInt` in pointer `myPtr`
- Indirection/dereferencing operator (*):

```
*myPtr = *myPtr * *myPtr;
```

  - Squares the value to which `myPtr` points

7.4 Calling Functions by Reference

Call-by-reference with pointer arguments

- Pass address of argument using address operator (&)
- Allows you to change the value at the specific location in memory
- Arrays are not passed with &, because the array name is already an adress/reference (pointer)

∗ operator

```c
void Double( int *myPointer )
 {
    *myPointer = 2 * ( *myPointer );
 }
```

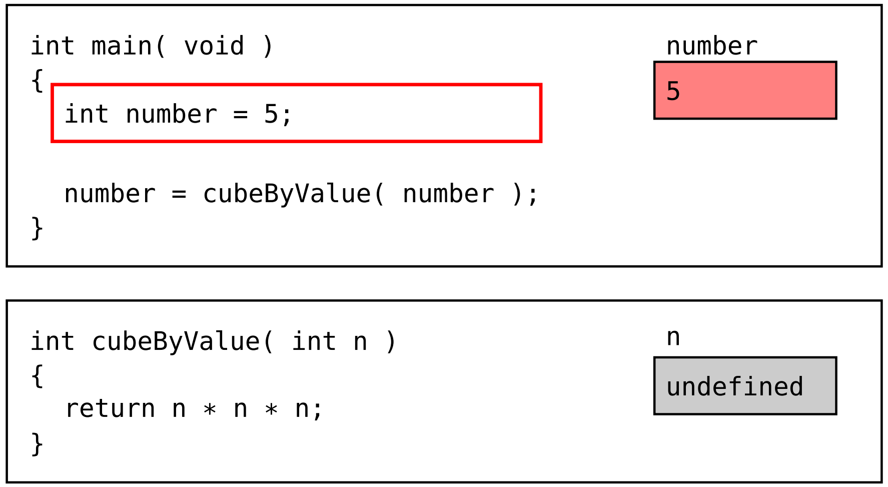
- The value of myPointer is the address of the data. myPointer "points" to the data

```c
//Fig 7.6: Cube a variable using call-by-value
#include <stdio.h>

int cubeByValue( int n ); // prototype

int main( void )
{
   int number = 5; // initialise number
   printf( "The original value of number is %d", number );

   // pass number by value to cubeByValue
   number = cubeByValue( number );
   printf( "\nThe new value of number is %d\n", number );
   return 0; // indicates successful termination
} // end main

// calculate and return cube of integer argument
int cubeByValue( int n )
{
   return n * n * n; // cube local variable n and return result
} // end function cubeByValue
```

Output

```
The original value of number is 5
The new value of number is 125
```

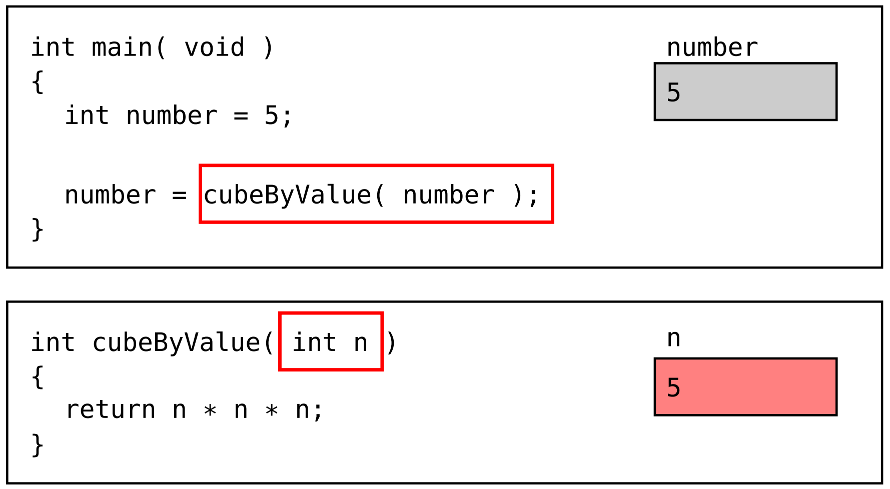
Visualisation of Call-by-value



```c
int main( void )
{
  int number = 5;

  number = cubeByValue( number );
}
```

```c
int cubeByValue( int n )
{
  return n * n * n;
}
```

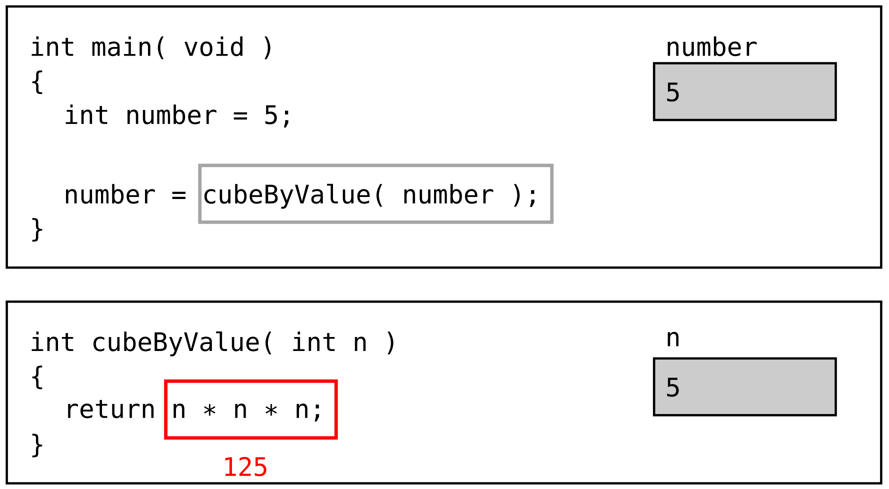
Visualisation of Call-by-value



```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

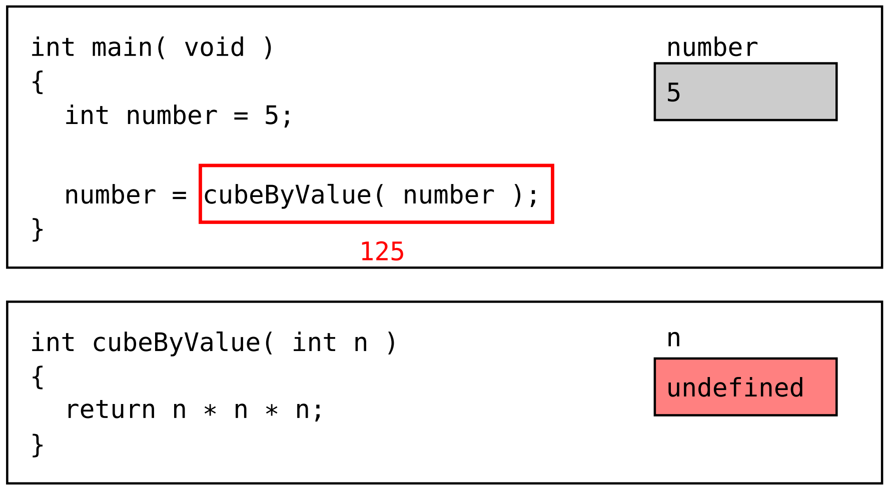
Visualisation of Call-by-value



```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

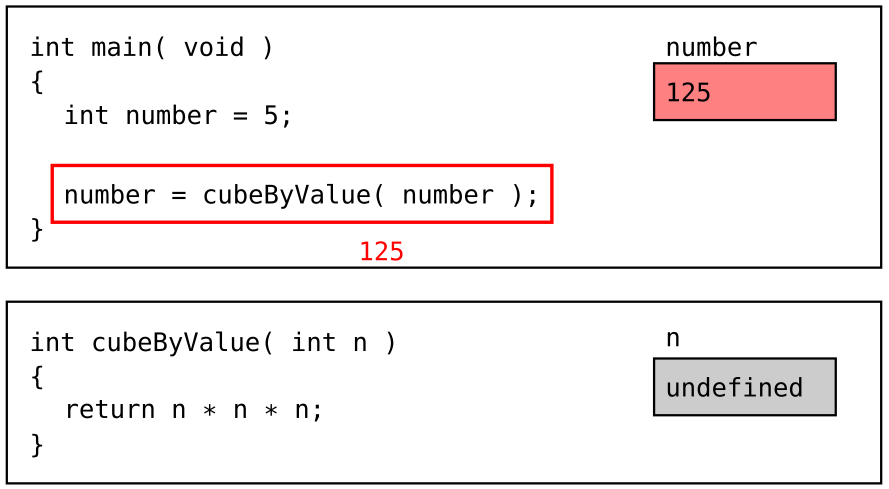
Visualisation of Call-by-value



```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

```
int cubeByValue( int n )
{
   return n * n * n;
}
```

Visualisation of Call-by-value



```
int main( void )
{
   int number = 5;

   number = cubeByValue( number );
}
```

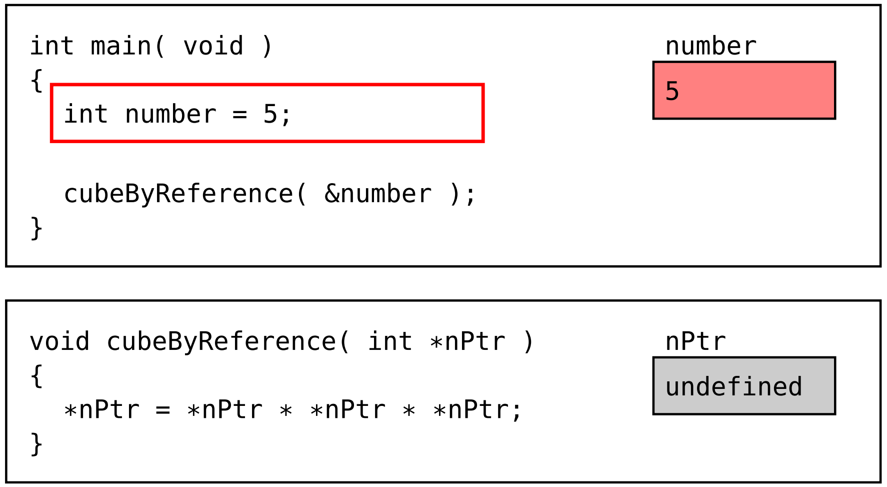
```
int cubeByValue( int n )
{
   return n * n * n;
}
```

```c
//Fig 7.7: Cube a variable using call-by-reference with pointer argument
#include <stdio.h>

void cubeByReference( int *nPtr ); // prototype

int main( void )
{
   int number = 5; // initialise number
   printf( "The original value of number is %d", number );

   // pass address of number to cubeByReference
   cubeByReference( &number );
   printf( "\nThe new value of number is %d\n", number );
   return 0; // indicates successful termination
} // end main

// calculate cube of *nPtr; modifies variable number in main
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr; // cube *nPtr
} // end function cubeByReference
```

Output

```
The original value of number is 5
The new value of number is 125
```

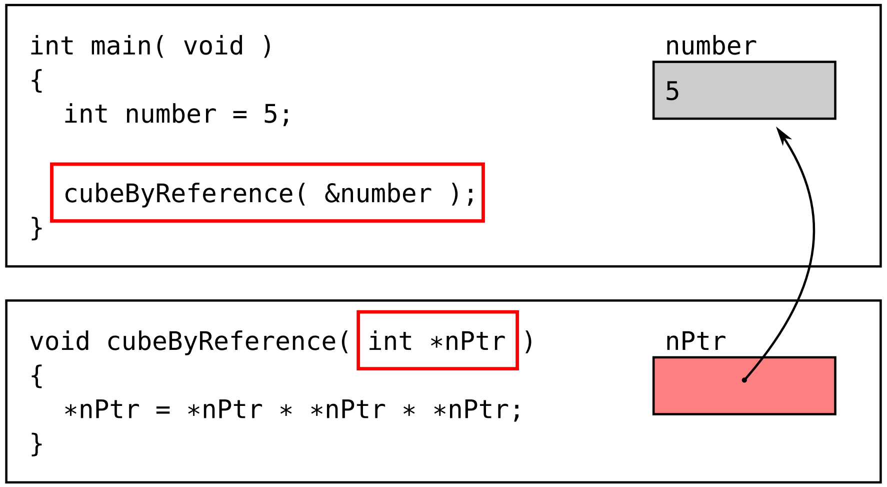
Visualisation of Call-by-reference



```c
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```c
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

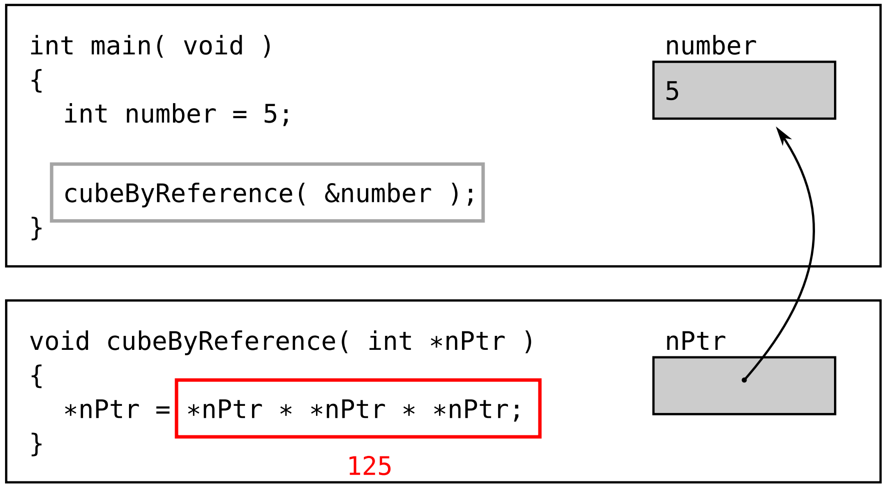
Visualisation of Call-by-reference



```c
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```c
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

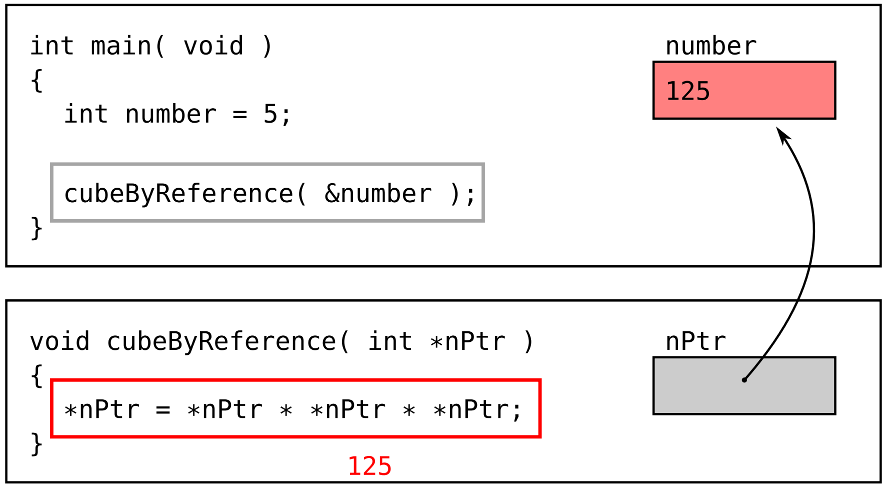
Visualisation of Call-by-reference



```
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

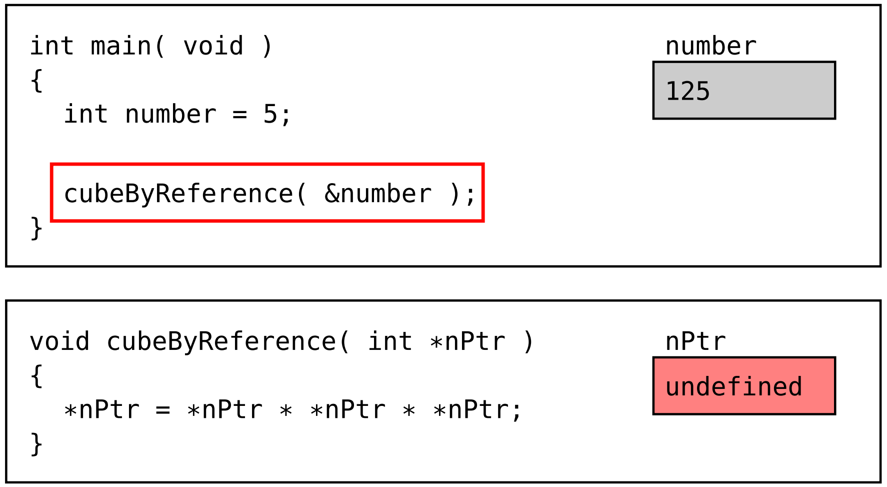
Visualisation of Call-by-reference



```
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

Visualisation of Call-by-reference



```
int main( void )
{
   int number = 5;

   cubeByReference( &number );
}
```

```
void cubeByReference( int *nPtr )
{
   *nPtr = *nPtr * *nPtr * *nPtr;
}
```

```c
//Swapping two integers using call-by-reference
#include <stdio.h>

void swap( int *a, int *b );

int main( void )
{
   int x = 7, y = -2; // declare and initialise 2 integers
   printf( "x = %d, y = %d\n", x, y );

   swap( &x, &y );    // swap 2 integers (call-by-reference)
   printf( "x = %d, y = %d\n", x, y );

   return 0; // indicates successful termination
} // end function main

void swap( int *a, int *b )
{
    int temp = *a;
    *a = *b;
    *b = temp;
} // end function swap
```

Output

```
x = 7, y = -2
x = -2, y = 7
```

Perspective

Today

Pointers II

- Passing pointers to functions

Next lecture

Discussion of test

Homework

1. Study Section 7.4 in Deitel & Deitel
2. Do Self Review Exercises 7.4, 7.5(a),(b) in Deitel & Deitel
3. Do Exercises 7.10 in Deitel & Deitel