COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

School of Computing
(University of Plymouth)

Outline

- C functions
- C preprocessor basic directives
- Measure the execution time of blocks of code
- Compare Floating Point numbers
- Structs in C
- Get an idea of what a pointer is (you learned this into assembly programming)
- Compilation of C programs

C routines (an example)

```
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main () {

   // local variable definition
   int a = 9;
   int b = 3;
   int output;

   /* calling the max function  */
   output = max(a, b);

   printf( "Max value is : %d\n", output );

   return 0;
}
```

```
/* definition of max function  */
int max(int num1, int num2) {

   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

C Preprocessor

- The preprocessor is an important part of C/C++ programming language.
- Before the C/C++ compiler compiles the code, the preprocessor makes a pass through the program looking for preprocessor directives.
- The output of this preprocessor step is then processed by the C/C++ compiler.
- All preprocessor commands begin with a hash symbol (#).

Most commonly used pre-processor directives

- The two most commonly used preprocessor directives are the following
  - **#include:** brings in code from another file, typically a header file or library, e.g., # include <stdio.h> loads the code from this library.
  - **# define:** allows a constant value to be declared for use throughout your code. Macro definitions are not variables and cannot be changed by your program code like variables.

1st Activity - Task1

- Check the notes

Measuring the execution time of a block of code

- C/C++ provide us with several functions being able to measure the execution time of a block of code, aka timers
- The code below is not enough, we need to run the function many times and take average time, why?
  - apart from our process, other processes use the hardware resources (e.g., cache, CPU) too.

```
start_time = timer();
Function();
end_time = timer();
Print(end_time-start_time);
```

2nd Activity - Tasks 2-4

□ Check the notes

Floating Point arithmetic

- Squeezing infinitely many real numbers into a finite number of bits requires an approximate representation
- As you have already been taught, 0.1+0.2 does not make 0.3 in most programming languages.
  - computers cannot accurately represent these numbers
- how can we compare two FP numbers?

```
If ( abs(( a-b) / b) ) < specific.value
//the two numbers are considered equal
else
//the two numbers are not considered equal
```

3rd Activity - Task 5

□ Check the notes

Structs

- A structure creates a data type that can be used to group items of possibly different types into a single type.

```
struct Datetime //this is how a struct is declared. Structure members cannot be
initialized here (when a datatype is declared, no memory is allocated for it.
Memory is allocated only when variables are created.). {
   unsigned short int year;
   char month[10];
   char day[10];
   unsigned short int hours;
   unsigned short int minutes;
   unsigned short int seconds;
};
```

4th Activity - Task 6

- See notes

What is a pointer?

- Pointers is an advanced topic.

- However, there are many C library functions that their operands are pointers and therefore you need to understand what they are.

- **You will not write any programs with pointers. You just need to understand how they work.**

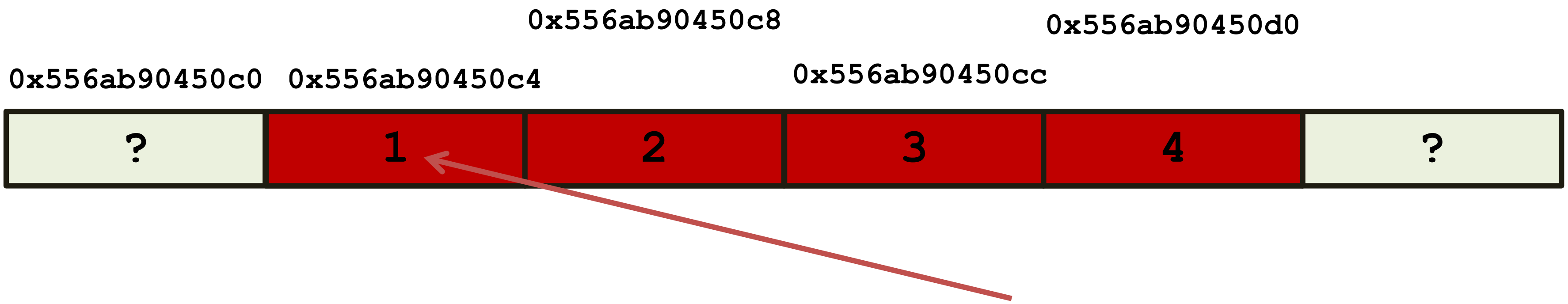
Pointers

- Pointers are an essential part of C / C++ programming.
- *A pointer is just a memory address*
- A pointer is a variable whose value is the address of another variable
  - It declared like any other variable : int * ptr;
  - The above is a pointer to an integer
- **Dereferencing**
  - If you put a * before a pointer, it means "***treat this pointer as if it were the actual variable***".
  - This is useful for changing the value at a memory address, instead of accessing a field

```
int var1 = 10;
int* ptr = &var1; //store address of var1 in pointer variable
*ptr += 10; // Same as "var1 += 10;"
```

Arrays are pointers (1)

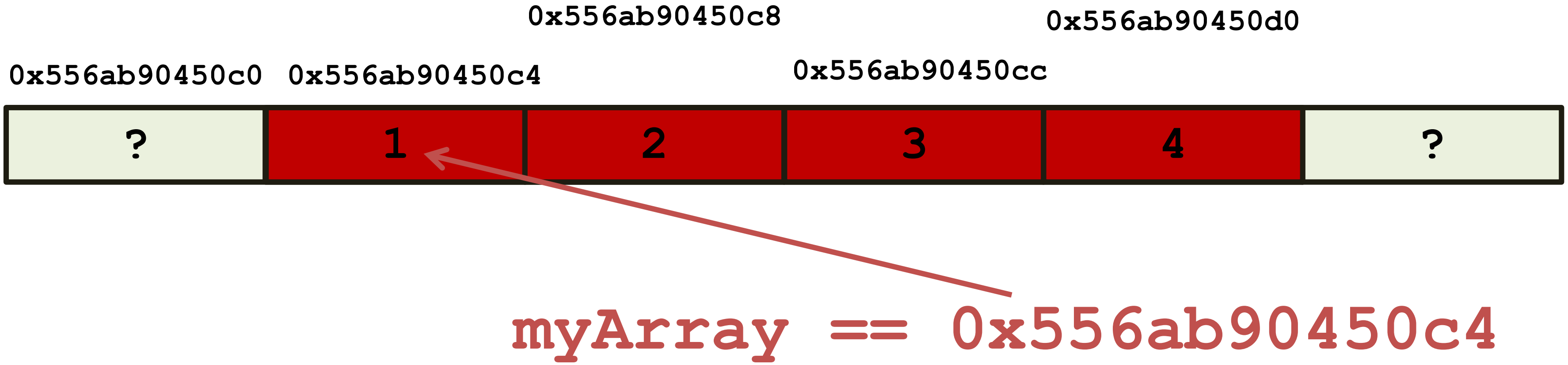
- Let's say we create an array of four integers called myArray:
  - int myArray[4] = {1 ,2 ,3 , 4} ;
- This creates space in memory for the four integers (16 bytes in total)
- … **and myArray is a pointer to the first element.**



`myArray == 0x556ab90450c4`

Arrays are pointers (2)

- The expression “myArray[i]” means “the value at memory address myArray + i*sizeof(int)”
- So “myArray[0] = 1;” means “copy the value 1 to memory address myArray”.
- myArray[1] = 2; means “copy the value 2 to memory address myArray+4”



0x556ab90450c8

0x556ab90450d0

0x556ab90450c0

0x556ab90450c4

0x556ab90450cc

?	1	2	3	4	?

myArray == 0x556ab90450c4

Pointers and one dimensional arrays (not assessed)

```
for (i = 0; i < N; i++) {
   printf("%d ", A[i] );
    }
```

```
 int *ptr = &A[0];
for (i = 0; i < N; i++) {
   printf("%d ", *(ptr + i) );
    }
```

```
for (i = 0; i < N; i++) {
   printf("%d ", *(A+i) );
    }
```

The above three codes are equivalent

Compiling and Running C programs
pre-processor

- **Step1**: The **pre-processor** processes the input code and stores it into the disc
  - The pre-processor commands start with '**#**'
  - ***#include <stdio.h>*** will force the pre-processor to copy paste the contents of the library into the source code at this location
  - '***#define N 100***' will force the pre-processor to copy paste the value of 100 wherever N is.

  - Keep in mind that there are two kinds of files
    - .c files contain function definitions
    - .h files contain function declarations

Compiling and Running C programs
compiler

- **Step2:** the **compiler** creates the object files and stores them into the disc
  - After the pre-processor has created another file including all the .c and .h files and expanding the #define and #include statements, the compiler compiles the program
  - **The source code is turned into *object files***
  - ***Object files contain the function definitions in binary form*** (.o files in linux or .obj in windows).
  - Object files are not executable

Compiling and Running C programs
linker and loader

- □ **Step3**: The **linker** links the object files with the appropriate libraries and creates an ***executable file***

- □ **Step4**: The loader puts the program into memory
  - Source code into memory
  - Data into memory

- □ **Step5**: The CPU executes the program

Further Reading

- If you want to learn more about C programming you can study the following links:
  - Tim Bailey, 2005, An Introduction to the C Programming Language and Software Design, available at: http://www-personal.acfr.usyd.edu.au/tbailey/ctext/ctext.pdf
  - C examples, Programiz, available at https://www.programiz.com/c-programming/examples
  - C Programming examples with Output, Beginners book, available at https://beginnersbook.com/2015/02/simple-c-programs/
  - C Programming Tutorial, from tutorialspoint.com, available at https://www.unf.edu/~wkloster/2220/ppts/cprogramming_tutorial.pdf

Thank you