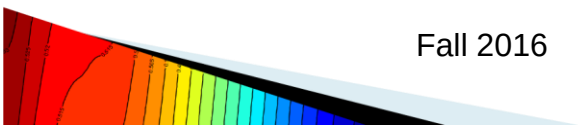
GNG 1106

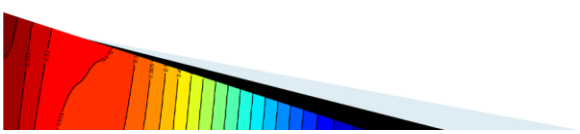
Fundamentals of Engineering Computation

Module 1 - Fundamental Concepts



Fall 2016

Topic 1: Introduction to the computer and computer languages



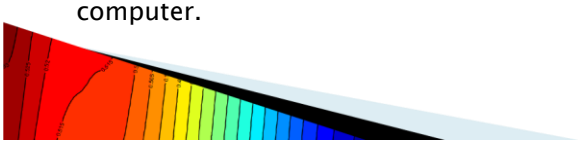
Introduction to computers

- What is a computer?
  - A computer is a device that is capable of making calculations and logical decisions at speeds 10^6 to 10^9 times faster than humans.
    - e.g.: A PC in 1996 could make about 10^7 additions/sec while a super-computer could make about 10^{11} additions/s.
  - A computer operates on data according to instructions or commands that are listed in a program. The program is written by a human (usually) using a programming language.
  - Both the computer (hardware device) and the program (software or commands) are required for a computer to accomplish a useful task.

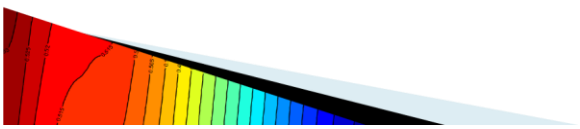
Computer Organization

The logical organization of most general purpose computers is almost always as follows:

- Input Unit.
  - Obtains information such as data and programs using devices such as a keyboard, diskettes, hard drives and network access cards.
- Output Unit.
  - Makes available to the outside world the results of computations using devices such as a screen, diskettes, hard drives and network access cards.
- Arithmetic Logic Unit (ALU).
  - Contains computing mechanisms to perform basic arithmetic operations such as addition, subtraction, multiplication and division, as well as the mechanisms required for logical decision making.
- Central Processing Unit (CPU).
  - Administrative section of the computer which coordinates the operation of the other sections. Synchronizes the operation of the computer.



- Memory Unit (RAM).
  - Has low storage capacity compared to a hard drive but is rapidly accessed.
  - Storage is temporary and data is lost when power to the memory unit is turned off.
  - Usually contains the programs in execution as well as the required input data and some output data.
- Secondary Storage Unit.
  - Long term, usually high capacity storage unit.
  - Access is slow relative to the memory unit.
  - Contains programs and data that are not in immediate use.

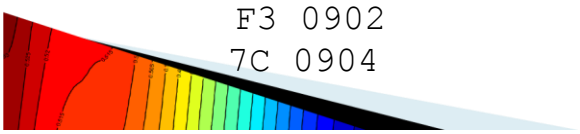


Programming Languages

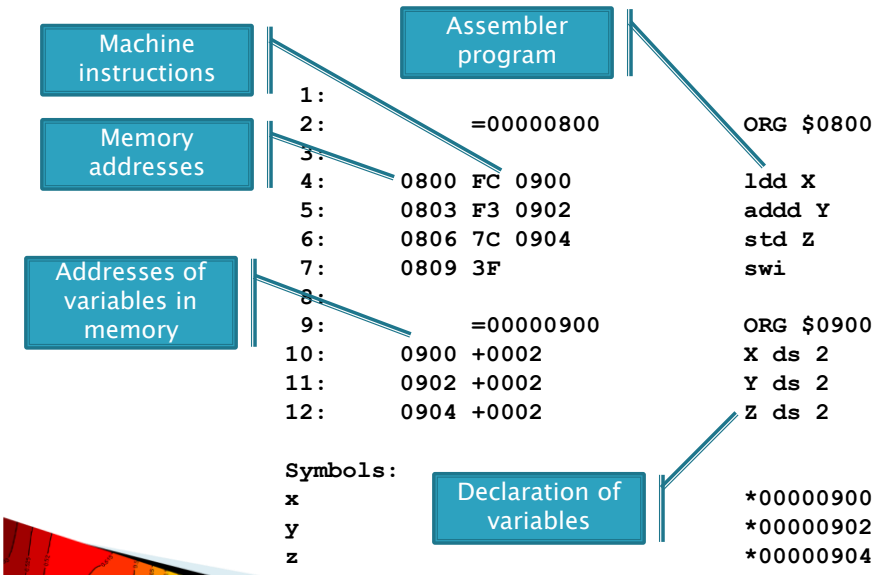
There exists three classifications or categories of programming languages.

- Machine language.
  - The only language that the computer hardware can understand directly.
  - All programs and data must be translated to machine language.
  - Language is machine dependent.
  - Machine language is generally difficult to manipulate since it consists essentially of numbers.
    - e.g.: Machine code for adding the content of two variables x and y and storing the result in variable z:

```
FC 0900
F3 0902
7C 0904
```



Assembly Program and Machine Instructions



- Assembly Language.
  - The machine language's numerical commands are associated to English-like abbreviations or mnemonics describing the command.
  - A program is written using the abbreviations and an assembler translates the program into machine language. Assembly language is still too cumbersome to manipulate for writing large and complex programs.
    - e.g.: Assembly code for adding the content of two variables x and y and storing the result in variable z:

```
LDD   x
ADDD  y
STD   z
```

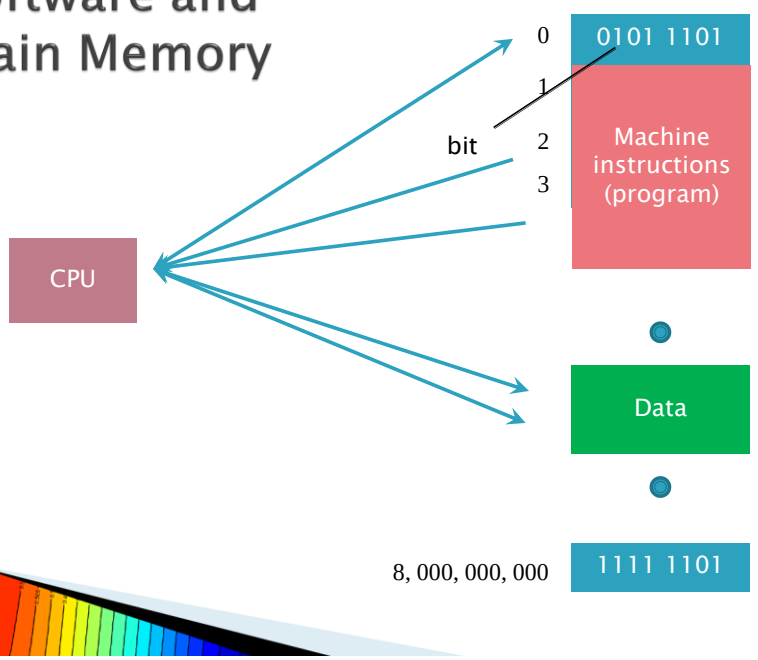
- High Level Languages.
  - Consists of a large number of powerful commands.
  - Commands are descriptive, mathematical and/or English-like.
  - A compiler translates the program written in high level language to a program in machine language.
    - e.g.: C code for adding the content of two variables x and y and storing the result in variable z:

```
z = x + y;
```

- Many high level programming languages have been created over the years. Some of the more commonly used general purpose high level languages include:
  - C: Created to help develop the UNIX operating system.
    - C remains quite popular with industry
  - FORTRAN: FORmula TRANslator. Commonly used for the development of applications for scientific or numerical computing.
  - Java: general purpose language that is object-oriented and has few implementation dependencies.
  - Python: is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java.

9

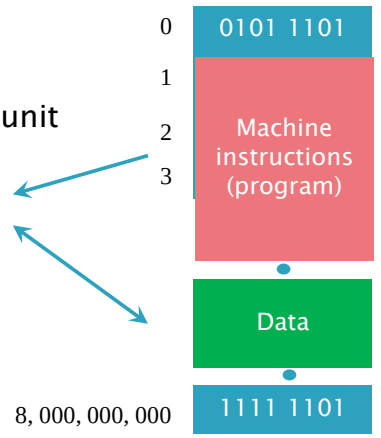
Software and Main Memory



10

Summary

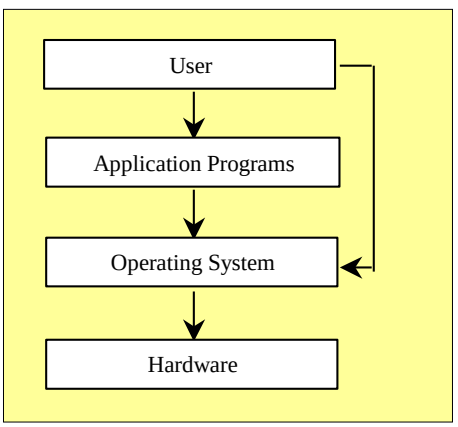
- Computer components
  - Input and output devices:
    - screen/window,
    - keyboard,
    - Hard disk
  - CPU– Central processing unit
  - Main memory
- Software
  - Runs in main memory
  - Instructions operate on values (data) in memory



11

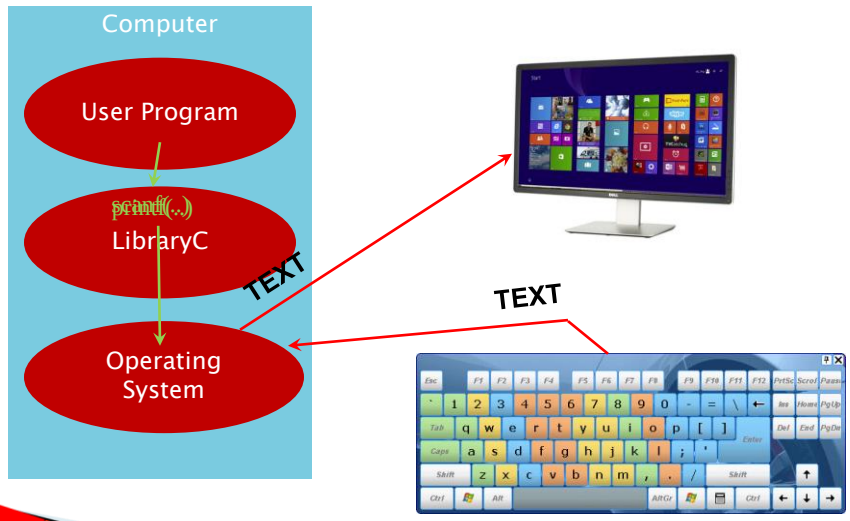
The role of Operating Systems

The *operating system* (OS) is a program that manages and controls a computer's activities. You are probably using Windows 98, NT, 2000, XP, or ME, Window 7, Windows 8. Windows is currently the most popular PC OS. Application programs such as an Internet browser and a word processor cannot run without an OS.



12

Operating System (OS)



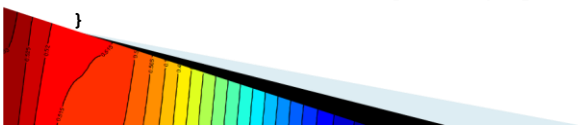
13

- The programming or coding process, using a high level language like C, involves many iterations of the following sequence:
  - **Editing** of the program using a text editor or word processor. An editor allows the programmer to write his/her code into electronic form and to save it in a computer file having a filename of the form: `program1.c`
  - **Compilation** of the code using the required language compiler. This translates the high level language program written by the programmer to machine language to create an executable file that can be loaded on the machine; the executable filename generally has the form: `program1.exe`. During the compilation, library code is added to the program code for form the executable file.
  - **Execution** of the program by the computer.
- **An Integrated Development Environment, the IDE, is a software application that supports the development of software using a high level language such as C. The IDE software allows to control each of the above steps.**
- **More on this during lab 1.**

14

A First C Program

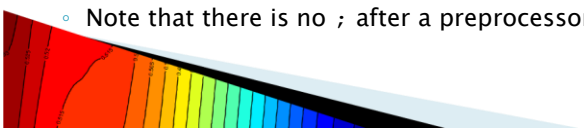
```
/*-------------------------------------------------------
File: welcome.c
Author: Gilbert Arbez (add your name to all programs you create)
Description: This section of comments (note the characters
             at the start and end of the comments) gives
             the name of the source code file and a
             short description of the program.  This
             program simply prints a short message on
             the console (screen).
-------------------------------------------------------*/
#include <stdio.h>
void main()
{
    printf("Welcome to GNG1106\n"); // standard function
                    // that requests a service from the
                    // operating system.
}
```



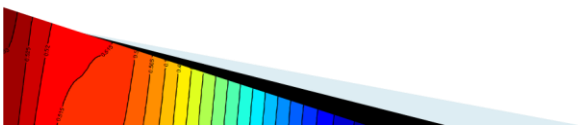
- Comment lines.
  - All lines that begin with `/*` and end with `*/` are comments.
  - Anything written between this pair is ignored by the compiler and does not generate any executable code.

    ```
    /* The comment is placed here. */
    ```
  - A comment may also be written using `//` placed anywhere on a line: anything written to the right of the `//` will be ignored by the compiler; this is actually a C++ command.

    ```
    // The comment is placed here.
    ```
  - Comments are written for humans by the programmer and are considered to be the internal documentation of the program; the external documentation is the software report associated with the program.
- The line `#include<stdio.h>` is a preprocessor directive.
  - All lines that begin with `#` are preprocessor directives which are processed before the compilation of the C code.
  - In this case, the directive `include` makes the compiler include a file named `stdio.h` into the compilation process. This file contains the standard C I/O (Input/Output) function headers and definitions.
  - Note that there is no `;` after a preprocessor directive.

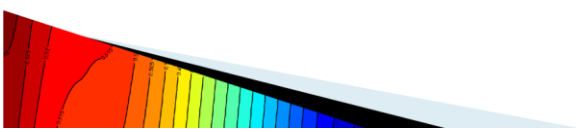


- Function header **`void`** `main()`.
  - The function name `main()` is required in all C programs.
  - All C programs begin executing after this line.
  - We shall study C function definitions in great detail; as we shall see, if `main()` is of type **`void`**, it returns nothing to the computer's operating system. If the parentheses `()` are empty, then the function does not receive arguments from the operating system.
- The left brace `{` indicates the beginning of instructions in `main()` and the right brace `}` indicates the end of the instructions.
  - This defines the instruction bloc for the function



17

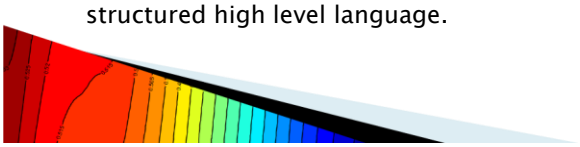
- The line `printf("Welcome\n");` is a call to the standard C I/O function `printf()` which prints out the desired message to the screen.
  - The message to be written is included in a pair of double quotes `"` and `"`.
  - `printf` is a function and the argument to the function is included in parentheses `()`.
  - `\n` is an escape sequence which positions the cursor at the beginning of the next line.
  - `printf` makes a call to the operating system
- The semi-colon `;` must end all C simple instructions.
- C is case sensitive.
  - UPPERCASE letters are not the same as lowercase letters.



18

Why C?

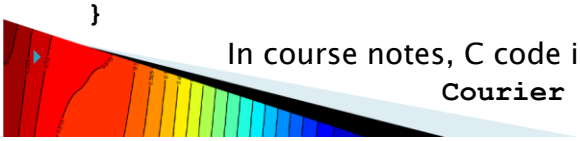
- C is a general purpose high level language.
- C is a structured language.
- C has some machine level commands. This makes it easy if a direct interaction with computer components is desired.
- C is still very popular with industry.
- We shall also study VBA at the end of the course.
- A structured language means that the language has the commands or constructs necessary for the development of modular programs.
  - A structured language will allow the programmer to create sub-programs (called *functions* in C) that perform a specific task like computing the hypotenuse of a right angle triangle.
  - Sub-programs help keep a large program readable and well-organized; furthermore, they can be re-used and shared.
  - As we shall see, there are other important features associated with a structured high level language.



C Instructions

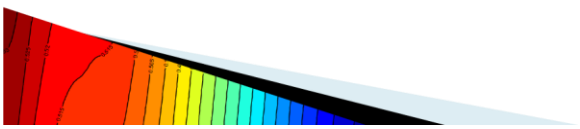
- C instructions can be :
  - A declaration of a variable: **int c;**
  - An arithmetic or logical expression: `a + b;`
  - An assignment expressions: `c = 1;`
  - A call to a function: `printf("a message\n");`
  - Complex instructions: decision and loop (shall see later).
- Instruction blocs :
  - A sequence of instructions enclosed in braces {}

```
{
        Instruction
        Instruction
        …
}
```



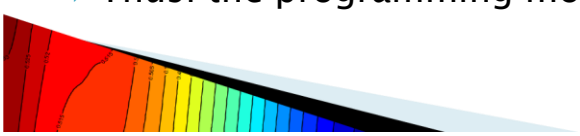
In course notes, C code is written in the **Courier New** font.

Topic 2: Introduction to the variable and calculations

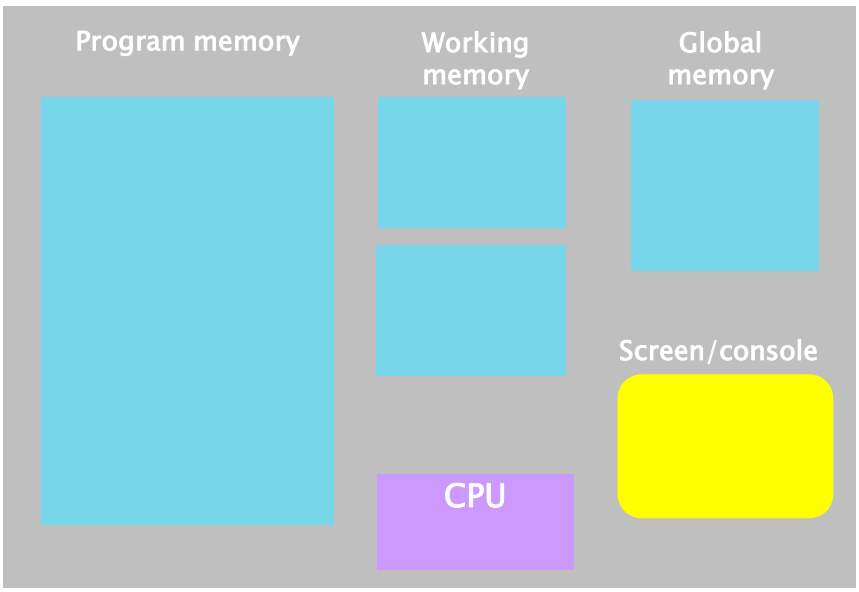


The challenge in computing

- How to understand the logic of a programming language and of software?
- One of the greatest challenge during the introduction of computing is to understand its logic. Research has shown that one of the greatest challenge for novice programmers is to develop a conceptual model of computing logic, i.e., of how the computer functions to interpret software.
- Thus: the programming model.



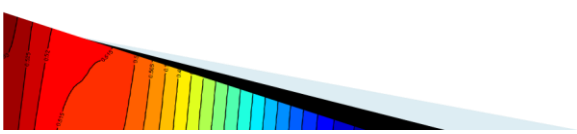
Programming model



23

Programming model (cont'd)

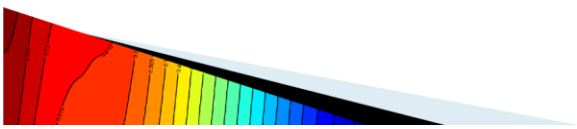
- **Program memory**
  Represents the part of main memory which contains the instructions executed by the CPU. Source code shall be placed in this part of the model.
- **Working memory**
  Represents the part of memory that contains data, i.e., computer variables. A part of working memory is reserved each time a function is executed.
- **Global memory**
  Represents the part of memory that contains data, i.e., computer variables. Variables declared outside of functions are found in this memory and are accessible by all functions in a program.



24

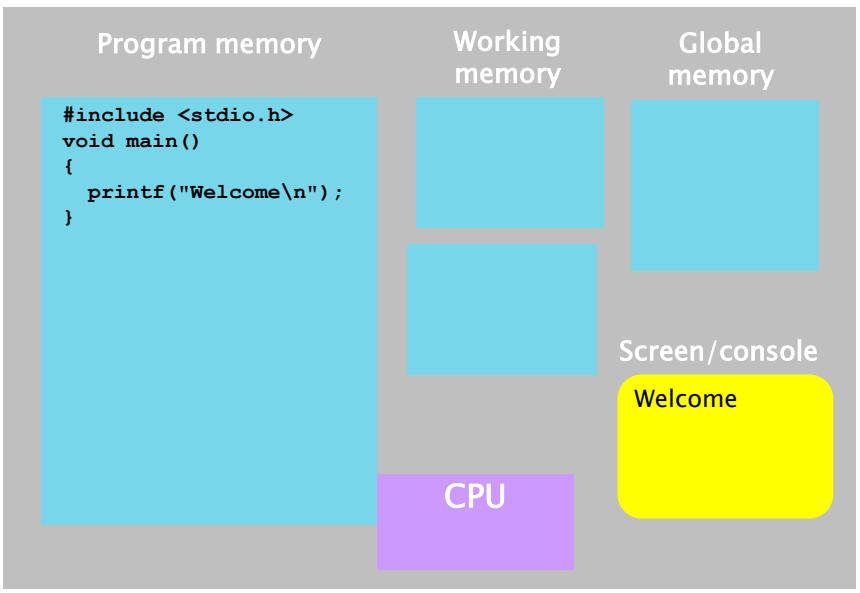
Programming model (cont'd)

- **CPU**
  The central processing unit is the computer component that executes program instructions. Although it is not part of main memory, it does contain a small internal memory used to load data values from memory to apply operations to the data using the ALU).
- **Screen/Console**
  The console represents a terminal or window in the computer terminal onto which a program can print text to interact with the user, for example, to request data and display results from calculations. Text typed in by the user and read in by the program are also shown on the console.



25

Programming model (example)



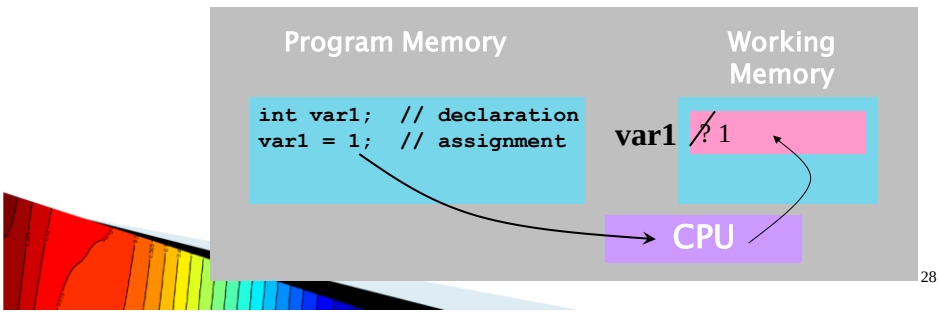
26

The computer variable

- The *variable* plays an important role in software, since it is the basic unit that contains data on which programs operate.
- Three characteristics to remember about a variable
  - Location in memory: contains a set of bits that represents its value.
  - Address: a number that determines where the variable is located in memory. The variable name corresponds to the address of the variable (when a program is compiled, a name is translated to an address for use by the CPU to access the contents of the variable
  - Type: how to interpret the contents, e.g. it takes less memory to store an integer value than a real value

Variable Declaration and Assignment of a Value

- A variable must be declared before being used in a program. The declaration is an instruction that reserves space for the variable.
  - Note that the value is unknown after it is declared (see ? below).
  - A declaration starts with its type followed by its name or many names separated by commas ",". It is terminated by a semi-colon ";".
- A value can be assigned to a variable, which means that a value is store in the location of memory reserved for the variable.
  - The content of a variable is changed using the assignment operator; the = is the assignment operator.
  - The assignment operator is destructive, i.e., the current value stored in the variable is replaced by the newly assigned value.
- Examine the following instructions: the first instruction creates the variable and the second assigns the value 1 to the variable.



28

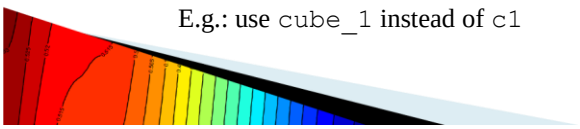
The name of a variable.

- All letters `a` to `z` and `A` to `Z`, all numbers `0` to `9` and the underscore _ can be used in the name of a variable.
- C is case sensitive so `a` is not recognized as being the same letter as `A`.

  E.g.: `CUBE_1` is not the same name as `cube_1`
- A variable name cannot begin with a number and names that begin with _ are often in bad programming style unless they are reserved for a specific set of variables.
- C programmers usually follow the convention that variable names are written in lowercase.
- Reserved words cannot be used as variable names.

  E.g.: `if`, `else`, `while`… are reserved words since they are used in C instructions so these words cannot be used as variable names.
- **Use descriptive variable names!**

  E.g.: use `cube_1` instead of `c1`

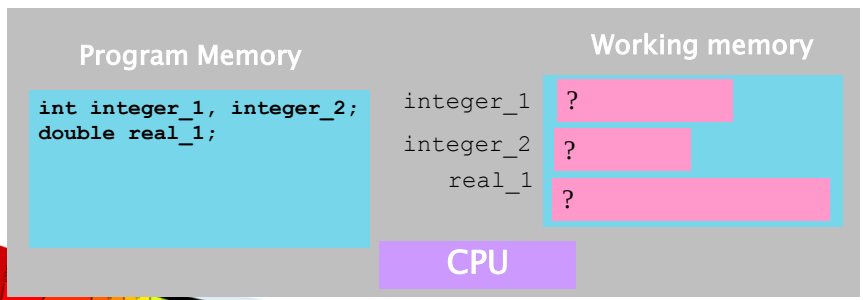


29

The type of a variable

- The type of a variable defines the nature or kind of data that can be stored in the variable.
- The following are two types used in C:
  - `int` to store integer values: `1`, `-1462`,…
  - `double` to store real values: `0.5`, `-2.34`,…

- Here are a few examples of variable declarations in C:
  - These declarations specify that the variables `integer_1` and `integer_2` are of type `int` and thus can store integer numbers, while `real_1` is a variable of type `float` that can store a real number. Space in memory will be allocated to these variables when the program is executed.

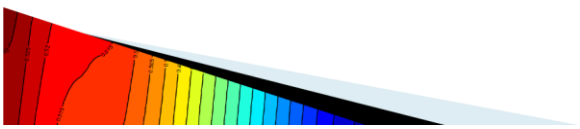


30

Calculations: arithmetic + assignment

- Familiar arithmetic expressions can be **evaluated** by a computer.
- But remember the definition of a variable in a computer.
- The CPU evaluates an expression to obtain a value
  - For example: **`x + y`**
  - The CPU gets the value stored in the variable x, then gets the value store in the variable y and adds the two values
  - What is done with the result of the addition?
- How is the following instruction executed:

  `z = x + y;  (remember the assignment operator)`

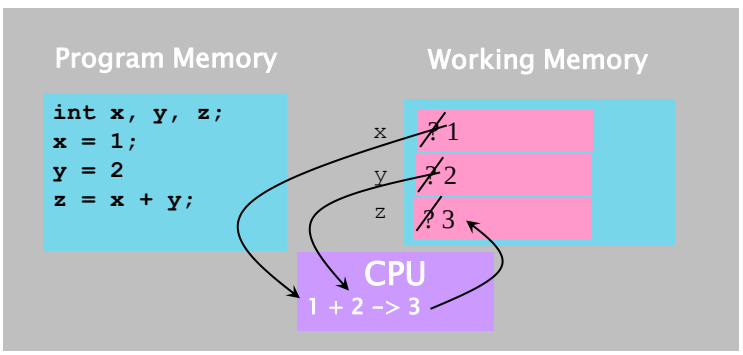


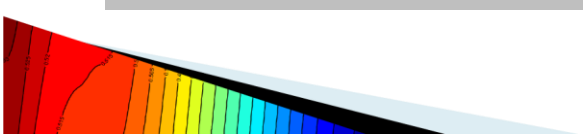
31

Calculations (continued)

- Examine how the CPU executes the instructions shown below
- How do you think the CPU executes the following instruction:

  `z = 1+(x+y)/2;`
- The computer does **one thing** at a time, but very fast.





32

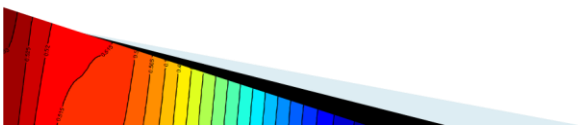
Arithmetic Operators

- The following binary arithmetic operators exist in C:

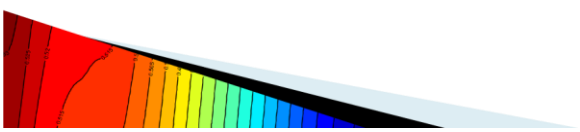
Operation	C Binary Operator	Typical C Arithmetic Sub-expression
addition	`+`	`a + b`
subtraction	`-`	`a - b`
multiplication	`*`	`a * b`
division	`/`	`a / b`

- The - operator can also be a unary operator when used to negate a variable.
- A standard C math function must be used for exponentiation: a^b
- Parentheses `()` are used to group sub-expressions in order to
  - make expressions more readable, and
  - change the order of precedence of the operators.

  E.g.: `e = (a + b) * (c + d);`



Topic 3: Basic Input/Output and the C Function



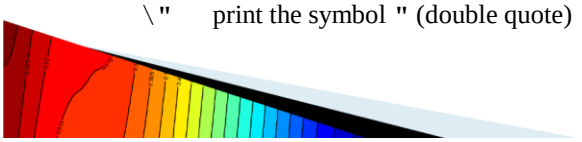
Introduction to output in C

- Data output to the screen is achieved using the standard C function **`printf`**.

  E.g.: **`printf("Message\n");`**

- This standard function will make the required calls to the operating system to print the desired message
- The message to be printed on the screen is enclosed in a pair of double-quotes " and ".
- The backslash **** in this context is called the escape character and **\n** is called an escape sequence. The escape sequence is an instruction to the **`printf`** function. There are many escape sequences:

\n	positions the cursor at the beginning of the next line
\t	move the cursor to the next tab position
\r	move the cursor to the beginning of the current line
\a	sound the bell (alarm)
\\	Print the symbol \ (backslash)
\"	print the symbol " (double quote)



Basic Output in C

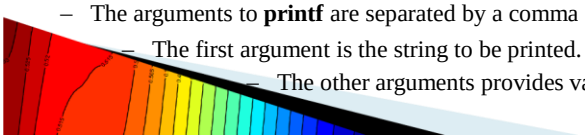
- **`printf`** can also be used to print out numbers and the content of variables to the screen:

```
E.g.:  printf("%d\n", 100);
       printf("%f\n", 101.3);
       printf("%d\n", integer_1);
       printf("%f\n", real_1);
```

- The `%d` is the conversion specifier used to print out an integer value and the `%f` is the conversion code to print a real value.
  - The conversion specifier informs **`printf`** of the type of the data to be printed.
  - A conversion specifier always begins with a `%`
  - The format **string** that contains the code specifiers must be enclosed in a pair of double quotes " and "
  - As we shall see, there are many more conversion specifiers. (To insert a % character in the string, use %%)
- The **`printf`** takes a number of arguments.

  E.g.: **`printf("The sum is %f\n", sum);`**

  - The arguments to **printf** are separated by a comma **,**
    - The first argument is the string to be printed.
    - The other arguments provides values for the conversion specifiers.



Basic Input

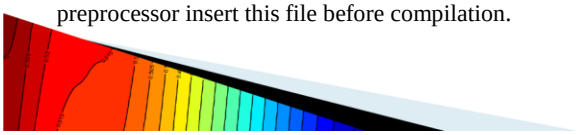
- Data can be read from the keyboard using the standard C function `scanf`.

E.g.:
```
scanf("%d", &integer_1);
scanf("%lf", &real_1); // real_1 is a double
scanf("%d%f", &integer_1, &real_1);
```

- `scanf` requires a conversion specifier that corresponds to the type of value to read.
- Note that **`%lf`** is required for **`scanf`** to read a value of type double (this differs from **`printf`**).
- An ampersand & must be placed before the variable name where the data value is to be stored; this indicates that the address of the variable is passed to the function so that the value can be stored in the variable's location in memory.

Standard I/O Header File

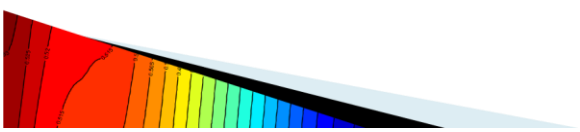
- Information about the standard functions `printf` and `scanf` is found in the standard I/O header file `stdio.h`.
- The preprocessor directive `#include <stdio.h>` placed before `main()` has the preprocessor insert this file before compilation.



The C Function

- In C, the subprogram is called a function.
- In C, a function is invoked via a function call. A function call begins with the name of the desired function followed by an argument list in parentheses. The argument list provides the data to be passed to the function.
  - E.g. `printf("Hello World.\n");`
  - In this example, the function `printf` is invoked through its name and the argument list contains only one argument to be passed which is the character string:

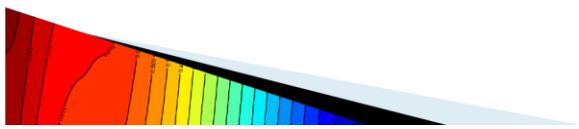
    `"Hello World.\n"`



Calling a Function

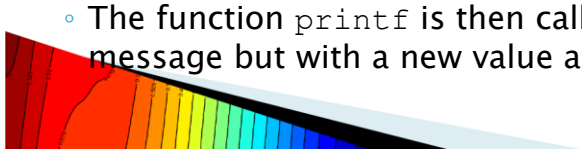
- The function, once called, executes the desired task with the given arguments and returns control to the caller when done.

E.g.:
```
#include <stdio.h>
void main()
{
    int num;
    num = 1;
    printf("Value of num: %d\n", num);
    num = 10;
    printf("Value of num: %d\n", num);
}
```



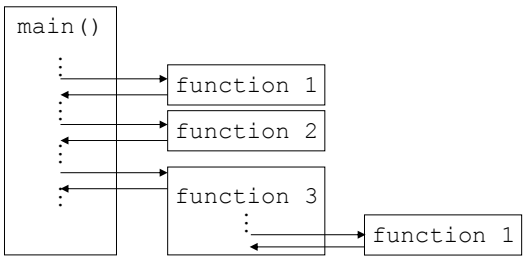
39

- In this example, `main` is the caller of the called function `printf`.
- Execution in `main` begins with the declaration of the variable `num`.
- The next instruction assigns `1` to the variable `num`.
- The function `printf` is then called with the arguments to be passed listed between parentheses.
  - Execution is now within the function `printf` which prints out the desired message to the screen.
- Once `printf` has completed its task, program execution resumes in `main`.
- Execution continues with the next instruction in `main`, that follows the call to `printf`, that is, the instruction which assigns `10` to the variable `num`.
- The function `printf` is then called again to print out a message but with a new value assigned to `num`.

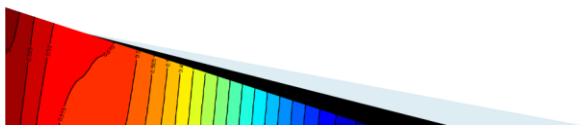


40

- Execution flow with many function calls:



- Functions can also return a value back to the caller.



41

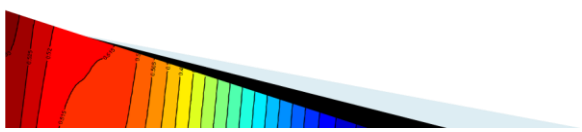
Definition of a Function

- It is possible to program your own functions in C; this is called function definition. The syntax of a function definition in C is:

```
type function_name(parameter list)
{
  <Instructions>
}
```

Careful: there is no ; here.

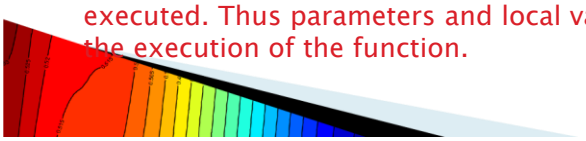
  - The function header (first line) describes the communication channel that will be established with the caller.
  - The name of the function is created from the same symbols as a variable name.
  - The type of the function identifies the type of the value returned by the function (e.g. `int` or `double`) .
    - If type is `void` then the function returns nothing to the caller.



42

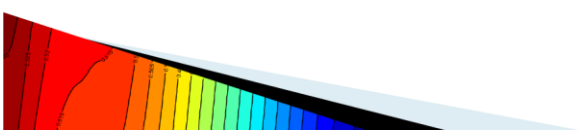
Function Parameters and Variables

- The parameter list contains the declaration of the variables to receive arguments passed to the function.
  - Individual declarations are separated by `,` (commas).
  - Parameters are also created in the function's working memory just like other declared variables.
  - The first argument in the function call is copied to the first parameter, the second argument is copied into the second parameter, and so on.
  - If the function does not receive arguments then the parameter list is of type `void`.
- All variable declarations and instructions in the function are placed between `{}`, i.e. an instruction bloc.
  - All variables declared in a function (including the parameters) are local; this means that they are not available and are not known outside of the function where they are defined.
  - Working memory is allocated to the function each time it is executed. Thus parameters and local variables exist only during the execution of the function.



Returning from a Function

- There are 3 ways of returning program control from a function back to the caller:
  1. Reaching the end of the function as delimited by the `}`
  2. By executing the command: `return;`
  3. By executing the command: `return expression;`
  - Mechanisms 1 or 2 are used in a function that does not return a value to the caller; **i.e.: functions of type `void`.**
  - Mechanism 3 must be used in a function that returns a value to the caller; `expression` must evaluate to a value having the same type as the function declaration.
    - This means that a function call can be used in any expression just like a variable
    - E.g. `z = 3 + sum(x, y);`
- A function cannot be defined within another function.



- Example: a simple function that prints out Hello World! to the screen.

```
void hello(void)
{
   printf("Hello World!\n");
}
```

This function receives no arguments and returns nothing to the caller.

- Example: a simple function that prints the sum of 2 integers.

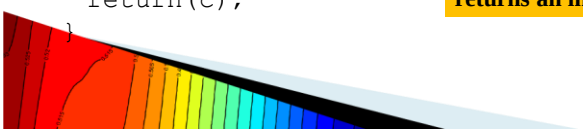
```
void prtsum(int a, int b)
{
  int c;
  c = a + b;
  printf("%d + %d = %d\n", a, b, c);
}
```

This function receives 2 arguments (integers) and returns nothing to the caller.

- Example: a simple function that returns sum of 2 integers to the caller.

```
int sum(int a, int b)
{
  int c;
  c = a + b;
  return(c);
}
```

This function receives 2 arguments (integers) and returns an integer to the caller.

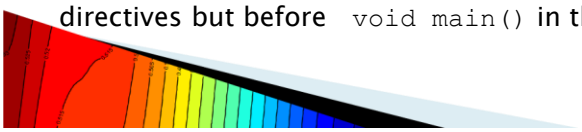


The Function Prototype

- The syntax of a function prototype is:
  - `type function_name(type, type, … , type);`

Careful: there is a ; here.

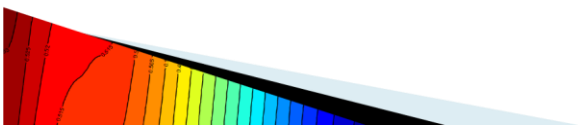
- The prototype:
  - declares that the function named `function_name`, will be called (e.g. by `main`).
  - defines the type of the value returned by the function,
  - identifies the number of parameters and the order of the types passed to the function.
- The prototype of a function must have:
  - the same name as in the function definition,
  - the same function type as in the definition,
  - the same number of parameters as in the definition, and
  - the same order of parameter types.
  - The parameter names are optional.
- The function prototypes are usually listed after the preprocessor directives but before `void main()` in the program file.



- The prototypes for our simple example functions are:

```
void hello(void);
void prtsum(int, int);
int sum (int, int);
```

- Heading files such as `stdio.h`, `math.h` and `stdlib.h` contain among other things, function prototypes for the standard C functions.
- Function prototypes are not required in a C program but you are strongly encouraged to include them for all functions called in the main program since they:
  - help others understand your program file,
  - can help avoid errors when calling the function.



47

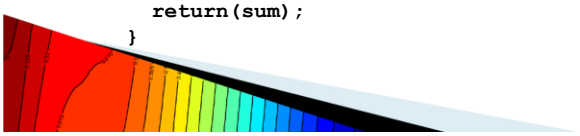
Put it All together!

```
#include <stdio.h>

int add2nums( int, int);

void main(void)
{
  int y,a,b;

  printf("Enter 2 numbers\n");
  scanf("%d%d", &a, &b);

  y = add2nums(a,b);

  printf("a is %d\n", a);
  printf("b is %d\n", b);
  printf("y is %d\n", y);
}

int add2nums( int num1, int num2)
{
  int sum;
  sum = num1 + num2;
  return(sum);
}
```

Function Prototype

Function call

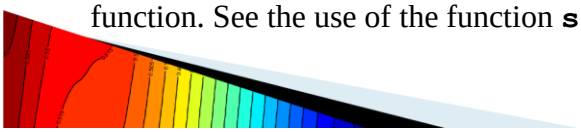
Function definition



48

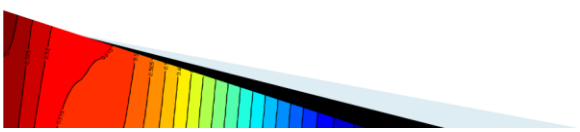
On the Passage of Arguments to a Function

- In many high level programming languages, there exists 2 ways of passing data to the function being called:
  - Pass by value.
  - Pass by reference.
- In a pass by value, the argument is a value is passed to the function (placed in a parameter); thus any source of the value remains unchanged in the caller even if the corresponding parameter is modified in the called function. Arguments can be expressions that are evaluated to values.
- In a pass by reference, the argument is an address to a value in memory passed to the function (placed in a parameter); thus the contents of the address (e.g. variable in the calling function) will be changed in the caller if the contents are modified in the called function. See the use of the function **`scanf`**.

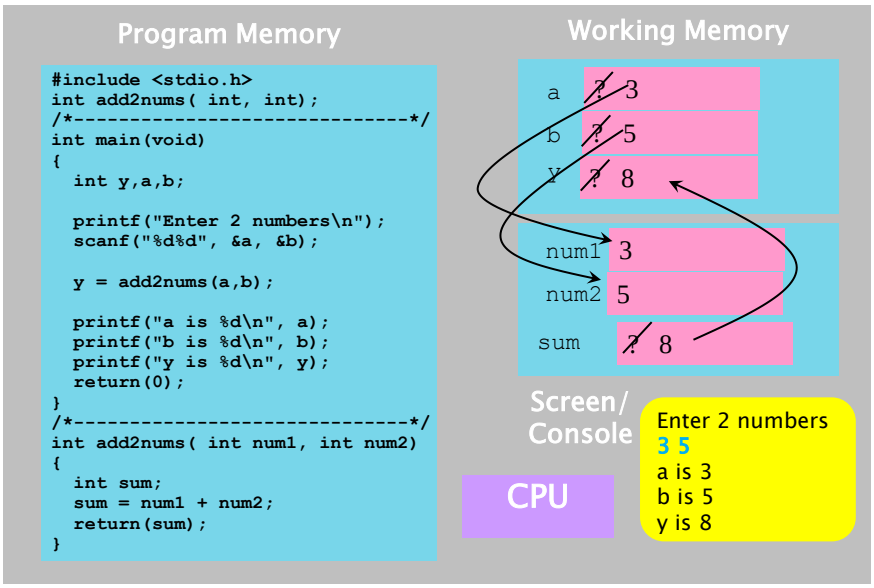


On the Passage of Arguments to a C Function

- **In C most arguments use pass by value**;
  - it is possible however to make a pass by reference using addressing operators (as we have seen with `scanf`) and
  - As we shall see later, addresses of arrays and strings are passed to functions.
- The rules of promotion are applied to mixed type arguments and to the type returned by the function.

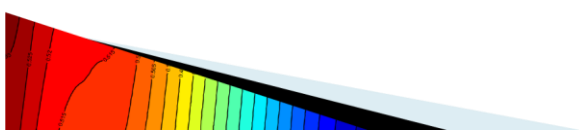


Example: execution of a function



GNG1106Template.c

- See the file *GNG1106.c*
- This will serve as a template for programs to be developed during the first half of the course.
- You shall be using it in your first lab.
- See the program in *introFunction.c* to see how the template was used to create a simple program.



Next Module

- Topic 1: Data Types and Simple Variables
- Topic 2: Arithmetic and Logic Expressions
- Topic 3: More on Input and Output
- Topic 4: Math Function Library