GNG 1106 Jan 11 Module 1

Download codeblocks!

For codeblocks:
- File
- New
- Project
- Console application
- C
- Then on left side… sources
- main .c

Introduction to computers:

What is a computer:
- A computer operates on data according to instructions or commands that are listed in a program. The program is typically written by a human using a programming language.
- Both the computer (hardware device) and the program (software and/ or commands) are needed for the computer to complete a useful task .

Computer organization:
- The logical organization of most general-purpose computers….and almost always is as follows:

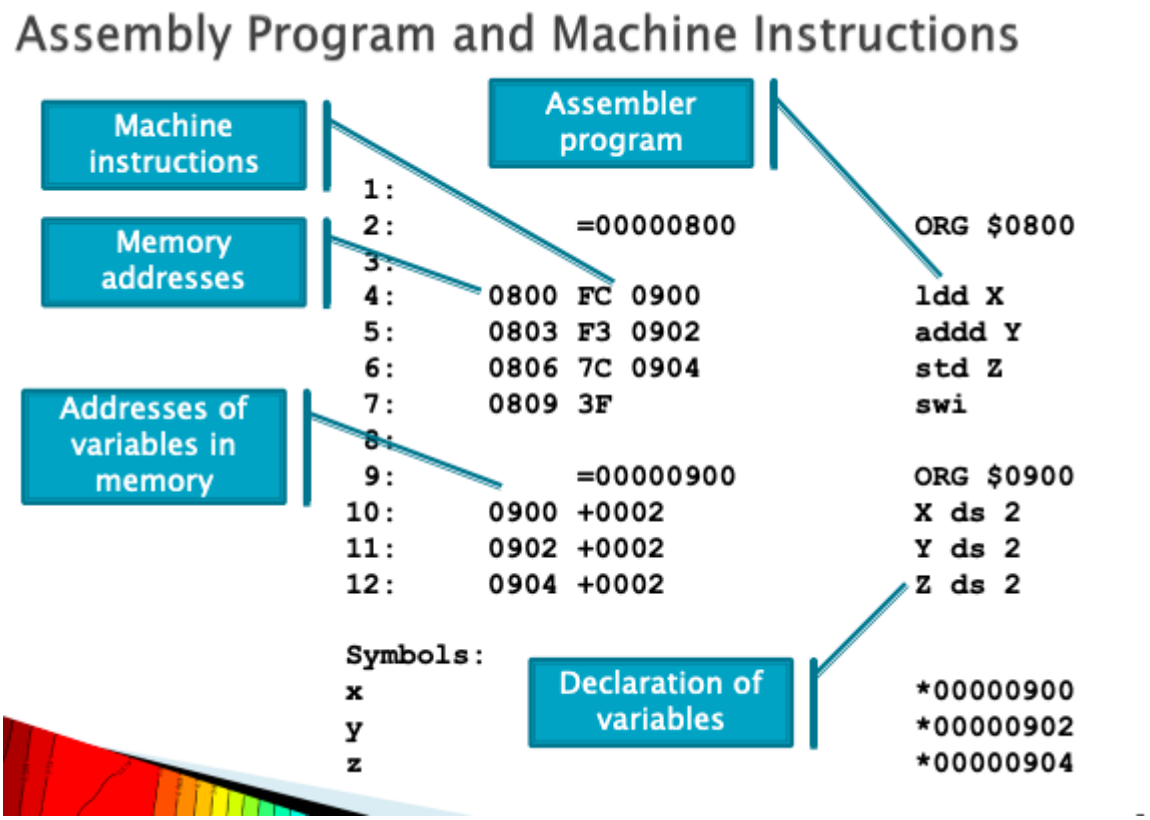
- Input unit: obtains information such as data and programs using devices suh as keyboard, harddrives, network access cards…
- Output unit: makes available to the outside world the results of the computations (presents the results in a useful way) such as screen, hard drives and network access cards.
- Arithmetic logic unit (ALU): contains computing mechanisms to perform basic arithmetic operations (math) such as addition, subtraction, multiplication and division, as well as mechanisms that are required for logical decision making
- Central processing unit (CPU): Administrative section of the computer which coordinates the operation of other sections…. Syncronizes the operation of the computer so that it is of use to a user.
- Memory Unit (RAM): This has low storage capacity compared to that of a harddrive, but is more rapidly accessed. This storage is temporary and the data may be lost when the power to the memory unit is turned off. The memory unit usually contains the programs in execution as well as the required input data and some output data.
- Secondary storage unit: long term, usually a high-capacity storage unit. Access to this unit is slow in relation to accessing the memory unit, and this unit contains programs and data that are not in immediate use.

Programming languages:
(there exist three classifications or categories of programming languages)
1. Machine language
    - The only language that the computer hardware can understand directly… all other programs and data must be translated into machine language to be used.

- Language is machine dependant (meaning it varies between systems)
- Machine language is typically difficult to manipulate since it consists almost solely of numbers

Ex. 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
```



2. Assembly language
    - The machine language's numerical commands are associated to english like abbreviations or mnemonics describing the function of the command.
    - A program is written using these abbreviations and an assembler translates the program into machine language.
    - Assembly language is still too cumbersome to manipulate for writing large and complex programs

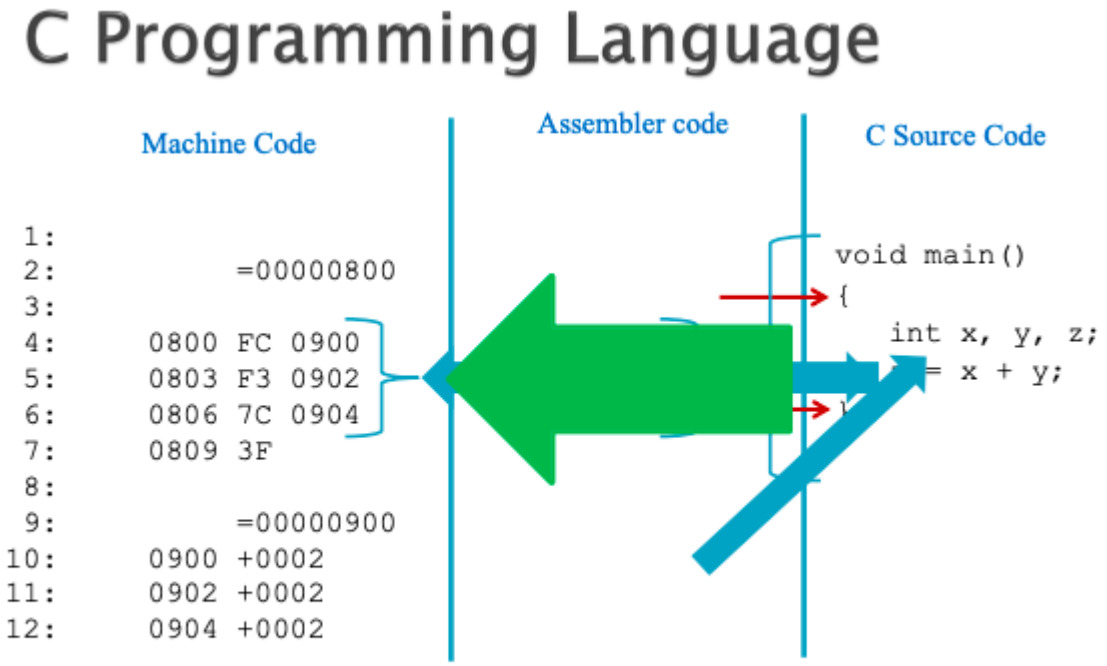
Ex. Assembly code for adding content of two variables x and y and storing the result in variable z:

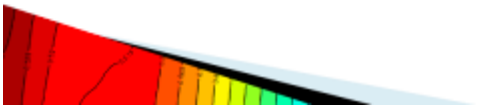
```
LDD   x
ADDD  Y
STD   z
```

3. High level languages:
    - Consist of many powerful commands
    - These commands are descriptive, mathematical or english like.
    - A compiler translates the program written in high level language to a program in machine language

Ex. C code for adding the content of two variables x and y and storing the result in z

Z = x + y





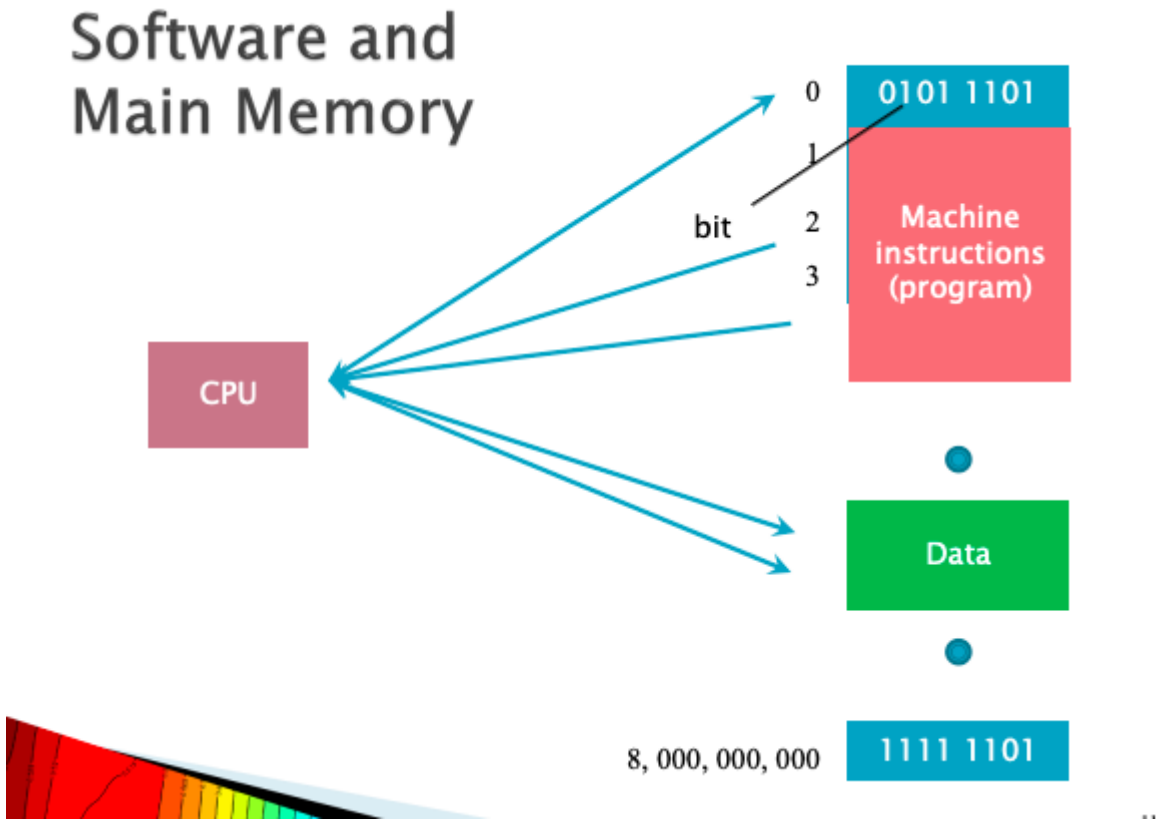
- There have been many high-level programming languages created over the years. Some of the more commonly used general-purpose-high-level languages include….

C: created to help develop the UNIX operating system. Remains one of the most popular languages in industry

FORTRAN: FORmula TRANslator…. Commonly used for the development of applications for scientific or numerical computing

JAVA: general purpose language. Java is object-oriented and has few implementation dependencies

Python: widely used general-purpose, high-level programming language. The design philosophy of python emphasizes code readability and its syntax allows programers to express concepts in fewer lines of code than would be possible in other languages such as C++ or Java.



Summary:

Computer components:

- Input and output devices:
  - Screen / window
  - Keyboard
  - Hard disk
- CPU - central processing unit
- Main memory

Software :

- Runs the main memory
- Instructions operate on values (data) in the memory

What is the an operating system?

- The operating system (os) is a program that manages and controls a computer's activities. (ex. Windows 7, Mac os….). Application programs such as an internet browser and a word processor cannot run without an operating system.

- THe programming or coding process, using a high-level language like c involves many iterations of eh following sequence…

1. Editing - editing the program using a text editor or word processor. An editor allows the programmer to write his or her code into electronic form and to save it in a computer file having a filename of the form: program1.c

2. Compilation - compilation of the code using the required language compiler. This translates the high-level language program that was written by the programmer into machine language to create and executable file that can be loaded on the machine… the executable filename generally takes the form: program1.exe… During the compilation, library code is added to the program code to form the executable file.

3. Execution - execution of the program by the computer.

Note: What is IDE?

An Integrated development environment, IDE, is a software application that supports the development of software using a high-level language such as c.
The IDE software allows you to control each of the above steps.

C Program:

Comment Lines: All lines that begin with /* and end with */ are comments. Anything written between this pair is ignored by the compiler and doesn't run in the code.

Ex. /* This is a comment */

Note: A comment may also be written by using // placed anywhere on a line: Anything written to the right of the // will be ignored by the compiler

Ex. // This is a comment

What is the purpose for comments: Comments are written for humans by the programmer and are considered to be internal documentation of the program… The external documentation is the software report associated with the program.

Processor directives: All Lines that begin with # are preprocessor directives which are processed before the compilation of the C code.

Ex. #include<stdio.h> ("include" makes the compiler include a file named "stdio.h" into the compilation process.)

Function header void: main ().
- The function name main () is required in all C programs
- All C programs begin executing after this line.
- Note: main () is type void, meaning it returns nothing to the computer's operating system.
- Note: if the parentheses () are empty, then the function does not receive arguments from the operating system.

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).

THe line printf(“Welcome\n”) ; is a call to the standard C I / O (central input output) function printf() which prints out the desired message to the screen.

- This message is written between a pair of double quotes “ “
- Printf is a function and the argument to the function is included in the parenthesis
- N\ is an escape sequence which positions the curser at the beginning of the next line
- Printf makes a call to the operating system

Note: the semi-colon ; must end all C simple instructions
Note: C is case sensitive (meaning that uppercase are not the same as lowercase)

Why use the C language:

- C is a general purpose high level language
- C is a structured language (meaning that the language has the commands or constructs necessary for the development of modular programs. Structured language will allow the programmer to create sub-programs called functions in C that perform a specific task) Note: sub-programs help keep a large program readable and well organized, they can also be reused and shared.
- C has some machine level commands, making it easily if a direct interaction with computer components is desired

C instructions:

- A declaration of a variable: int c;
- An arithmetic or logis expression: a + b;
- An assignment expression: c = 1;
- A call to a function: printf(“a message\n”);
- Or Complex instructions: decision and loop

What is an instruction bloc?

- A sequence of instructions enclosed in braces { }

Ex.

```
{
        Instruction
        Instruction
        …

}
```

<u>Introduction to the variable and calculations:</u>

Code 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 (ex. Computer variables). A part of the working memory is reserved each time a function is executed

Global memory:

- Represents the part of memory that contains data (ex. Computer variables). Variables that are declared outside of functions are found in this memory and are accessible by all functions in the program

CPU:

- The central processing unit is the computer component that executes program instructions. Although it isnt a part of the mian memory, it contains 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.

The computer variable:

- The variable plays an important role in software, since it is the basic unit that contains data on which programs operative.
- Location of a variable in the memory: contains a set of bits that represents its value
- Address of a variable: A number that determines where the variable is located in memory. The variable name corresponds to the address of the variable.
- Type of variable: how to interpret the contents (ex. 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 that variable. (Note: 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 stored in the location of memory reserved for the variable. (Note: the content of a variable is changed using the assignment operator “ = “... if the current value stored in the variable is replaced by a new assigned value, the previous value is erased.)

The name of a variable:

- All letters and numbers and the underscore “ _ “ can be used to name a variable
- Note that C language is case sensitive, so “a” is not recognized the same as “A”... further example: “CUBE_1” is not the same as “cube_1”
- A variable name cannot begin with a number and the names that begin with “ _ “ are considered bad programming style unless they are reserved for a specific set of variables.
- Note: typically, C programmers follow the convention that variable names are written in lower case
- Note: Reserved words cannot be used in variable names.
  Ex. the words… if, else, while… are reserved words since tehy are all used in C instructions, therefore they cannot be used in variable names.

Tip: it is better to use more descriptive variable names, this makes it clear to you and others trying to understand your code.

Types of variables:

- The type of variable defines the nature or kind of data that can be stored in that variable.
- Ex. int - used to store integer values
- Ex. double - used to store real values (ex. 0.5 or 35.34) Note that doubles are 64 bits
- Ex. float - used to store a real value…. Number with decimal (note that a float is 32 bits)

Calculations (Arithmetic and assignment):

- Familiar arithmetic expressions can be evaluated by a computer
- The CPU evaluates an expression to obtain a value.. Ex. x + y
- THe CPU gets the value stored in the variable x, then gets the value stored in the variable y and adds the two values.
- This result should then be stored somehow… for example, in a new variable that stores the result. ex . z = x + y

Arithmetic operators:

Operation	C binary operator	Typical C arithmetic sub expression
Addition	+	A + B
Subtraction	-	A - B
Multiplication	*	A * B
Division	/	A / B

Note: parentheses are used to group sub-expressions in order to make expressions more readable and change the order of precedence of the operators.

Ex. e = (a + b) * (c + d) ;

<u>Basic input / output and the C function:</u>

Introduction to output in C

- Data output to the screen is achieved using the standard C function “ printf “

  Ex. printf(“Message\n”) ;

- The message to be printed on the screen is enclosed in a pair of double quotes “ “/
- 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 instruction.

Types of escape sequences:

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

Printf can also be used to print out numbers and the content of variables to the screen:

Ex. prinf("%d\n", 100) ;
Ex. printf("%f\n", 101.3) ;
Ex. printf("%d\n", integer_1) ;
Ex. 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 data to be printed
    - Note: the conversion specifier always begins with %
    - The format "string" that contains the code specifiers must be enclosed in a pair of double quotes " and ".
    - Note: to insert a % character into a string… use %%

- Note: printf takes a number of arguments… These arguments are separated by a comma. The first argument is to be printed, the other arguments provide values for the conversion specifiers.

  Ex. printf("The sum is %f\n", sum) ;

Basic input:

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

Ex. scanf("%d", &integer_1) ;
Ex. scanf("%lf", &real_1) ; // real_is a double
Ex. scanf("%d%f", &integer_1, &real_1) ;

Note: scanf requires a conversion specifier that corresponds to the type of value to read.
Note: %lf is required for scanf to read a value of type double.
Note: an ampersand (&) must be placed before the variable name where the data 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 inthe 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 parenthesis. The argument list provides the data to be passed to the function.

  Ex. printf(“Hello World. \n”) ;

- In this example, the function printf is invoked through its name and the argument list contains onl one argument to be passed which is the character string: “Hello World. \n”

Note: 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);
}
```

- 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 listen 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 resumed 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”

Definition of a function

- It is possible to program your own functions in c, these are called function definition…

Note: 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 (ex. Int, double..)
- Note: if type is “void” then the function returns nothing to the caller.

Function parameters and variables:

- The parameterlist 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 functions working memory just like other declared variables
- The first argument inthe 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 { } (ie. an instruction bloc)

Note: all variables declared in a function (including the parameters) are local… meaning that they are not available and are not known outside of the function where they are defined.
Note: 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 three 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 (ie. the 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)

  Ex. z = 3 + sum (x , y) ;

Note: 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) ;

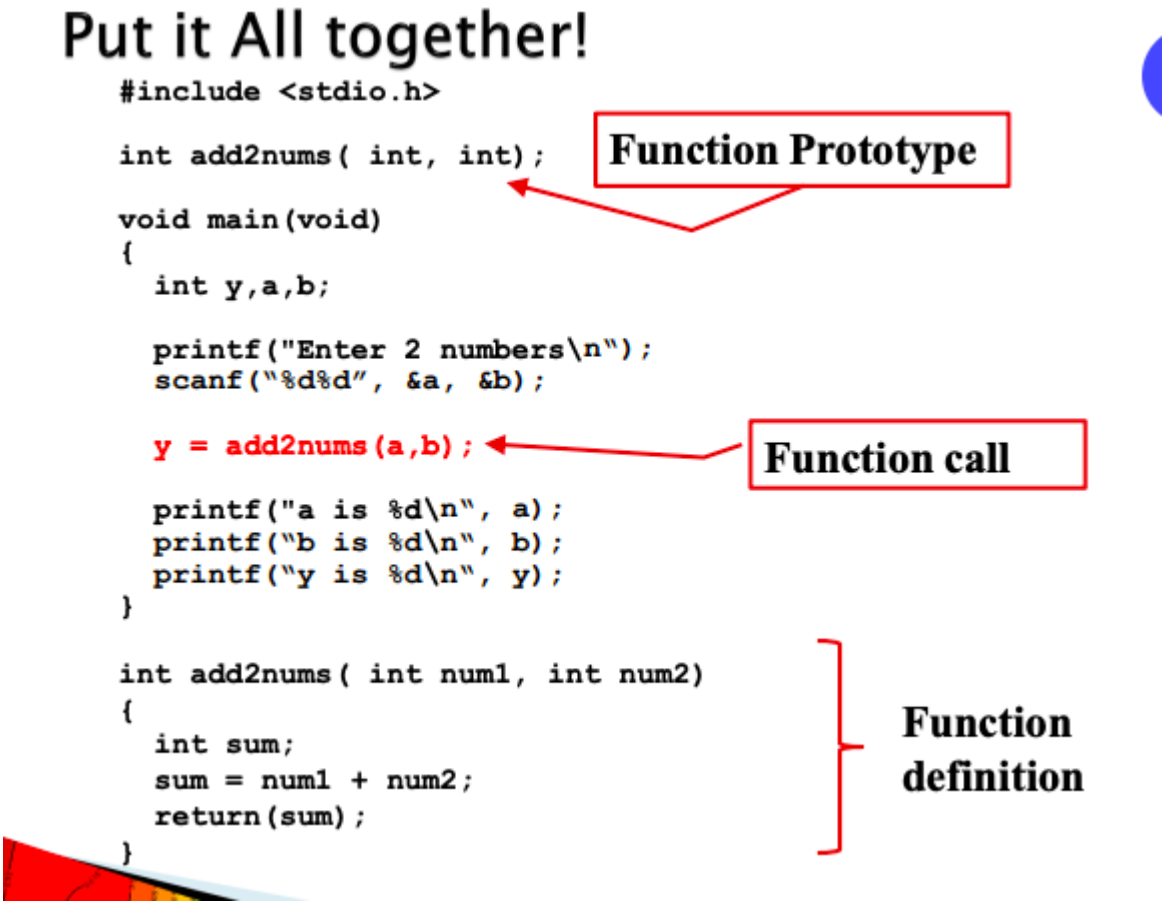
- The prototype declares that the function named function_name will be called (ex main)
- Defines the type of the value returned by the function
- Identifies the number of parameters and the order of the typed passed to the function
- Note: the prototype of a function must have
- 1. The same name as i the function definition
- 2. The same function type as in the definition
- 3. The sae number of parameters as in the definition
- 4. The same order of parameter types
- 5. The parameter names are optional
- Note: the function prototypes are usually listed after the preprocessor directives but before void main () in the program file.

Prototypes for simple example functions:

Void hello (void) ;
Void prtsum (int, int) ;
Int sum (int, int) ;

Note: heading files such as stdio.h, math.h and stdlib.h contain among other things, function prototypes for the standard C functions.

Note: function prototypes are not required in a C program but it is strongly encouraged that they are used for all functions called in the main program since they help others understand the program file and can help avoid errors when calling a function.



Put it All together – Execution of main()

- The execution of a program always starts with the main function.
- Instructions are executed one at a time

```
int y,a,b;    // Declares variables
printf("Enter 2 numbers\n");  // Prints string
scanf("%d%d", &a, &b);  // Reads 2 values from the
                        // keyboard and stores them in a and b
```

- Next instruction includes a call to add2nums (see next slide)

```
y = add2nums(a,b);  // Assigns return value to y
```

- Last 3 instructions

```
printf("a is %d\n", a);  // Prints string with value of a
printf("b is %d\n", b);  // Prints string with value of b
printf("y is %d\n", y);  // Prints string with value of y
```

Put it All together – Calling add2nums

- When a function is called, the execution of the calling function is suspended.
  - Execution of main at instruction `y = add2nums(a,b);` is suspended.
- The arguments in the function call are evaluated to values
  - The arguments **a,b** are evaluated to the values stored in the variables (see slide 53, **a,b** are evaluated to 3,5)
  - Thus when a variable is given as an argument, their value is passed to the function.
- A piece of working memory is allocated to the execution of the called function
  - Working memory is reserved for executing add2nums

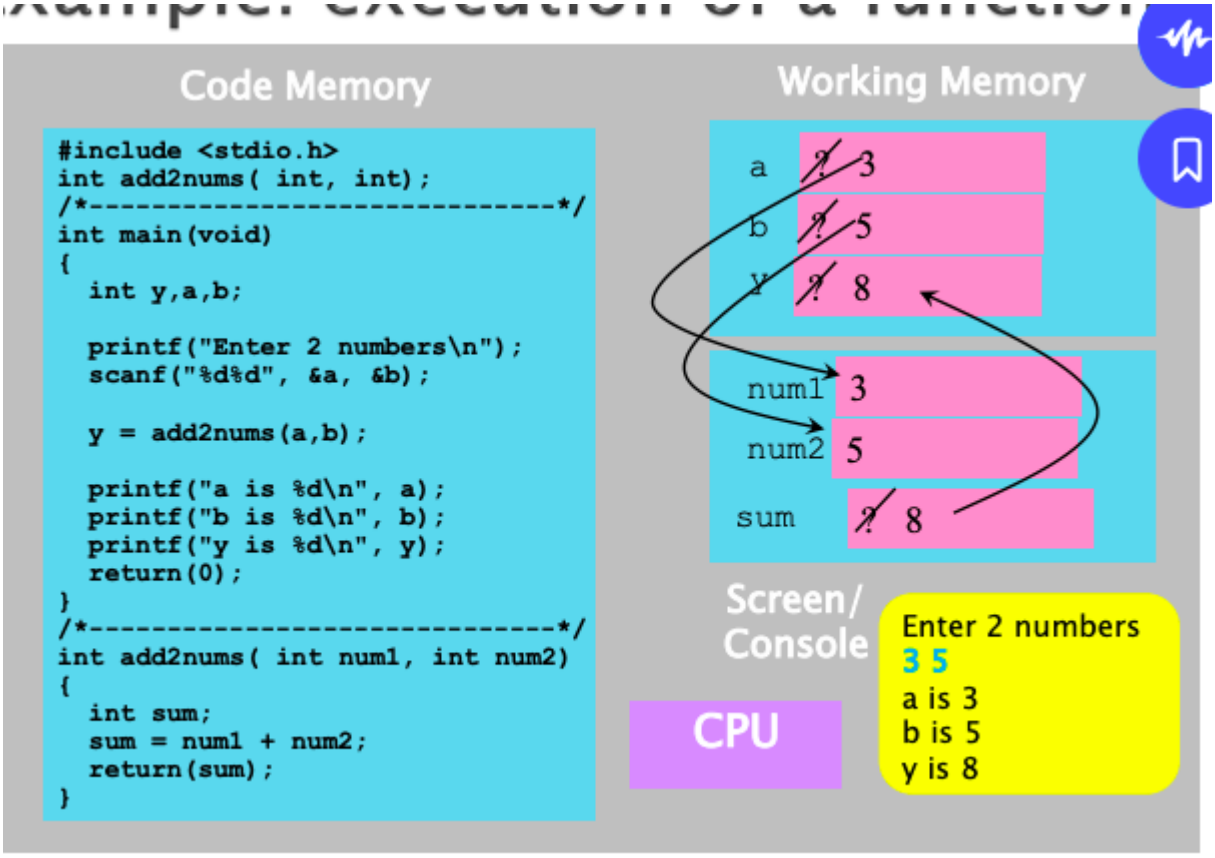
Put it All together – Calling add2nums

- The parameters are created in the working memory and the argument values are copied into the parameters
  - Parameters are thus variables, and the variables num1 and num2 are created in the working memory.
  - The values of the arguments (i.e. values of a, b) are copied into parameters num1, num2 respectively, that is according to the positions of the arguments/parameters (see next slide).
- The function instructions are then executed.

```
int sum;    // Declares variable – sum is created in W.M.
sum = num1 + num2;  // Values of num1/num2 summed and
                    // assigned to sum
return(sum);  // returns the value in sum
```

- At the end of the function, its execution terminates and "returns" a value. The calling resumes its execution with the returned value.
  - The allocated memory is retrieved (all variables are no longer accessible).
    - The CPU maintains the returned value to be used in the calling function.

Example: execution of a function



On the passage of arguments to a function:

- In many high level programming languages, there exist 2 ways of passing data to the function being called: 1. Pass by value 2. Pass by reference
- IN pass by value, the argument is a value is passed to the function (placed in a parameter); thus any soure 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 (eg. variable in the calling function) will be changed in the caller if the contents are modified in the called function.

Note: in C, most arguments use pass by value… it is possible however to make a pass by reference using addressing operators (as seen in scanf) and 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