INTRODUCTION

CS10003: PROGRAMMING AND DATA STRUCTURES

What is this course?

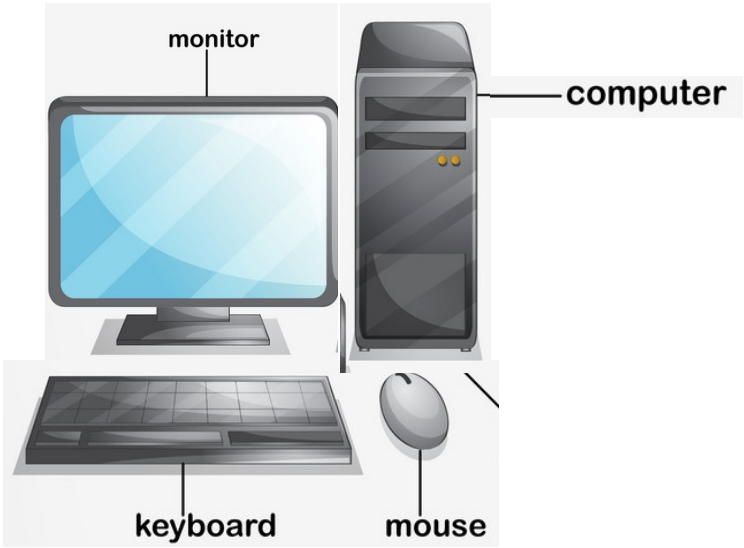
PDS: Programming and Data structure

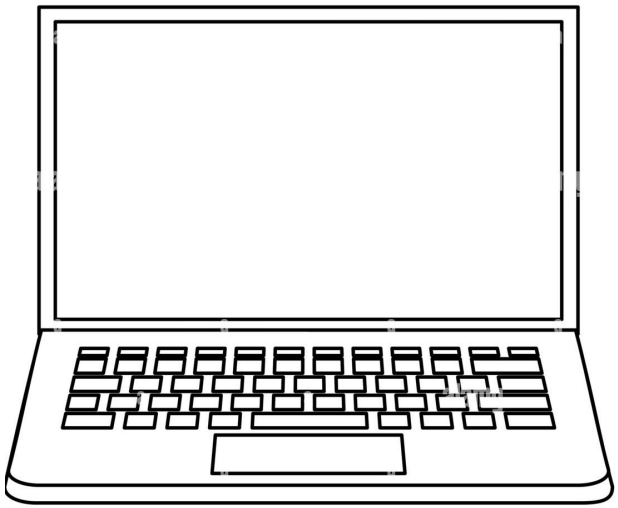
- Basically we will learn how to write programs
- Programs? Set of instructions written for a computer. Tell it what to do

What is this course?

PDS: Programming and Data structure

- **Basically we will learn how to write programs**
- **Programs? Set of instructions written for a computer. Tell it what to do**





General announcements about the course

This is the PDS theory course. There is a Lab course that will run in parallel.

Slides, announcements will be put up on the theory course website (you should check it frequently)

https://iitkgp-pds-autumn-2627.github.io/

Books:

1. **Programming with C, Byron Gottfried**
2. **The C Programming Language, Brian W Kernighan, Dennis M Ritchie**
3. **Programming in ANSI C, E. Balaguruswamy**
4. **Data Structures, S. Lipschutz, Schaum's Outline Series**

… and many more options

More materials available at http://cse.iitkgp.ac.in/pds/notes

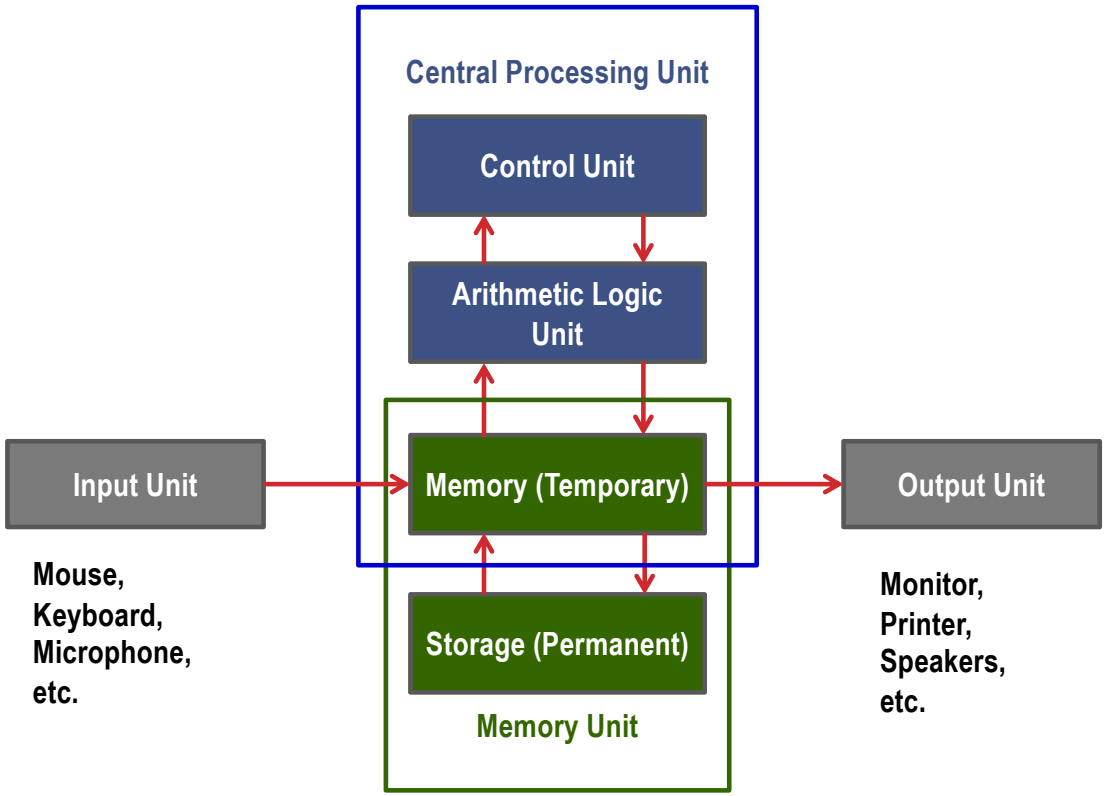
Evaluation Plan

- Two Class tests (1 hour)
  - Weightage: 30% of the total marks (15% each)
- Mid-semester exam
  - Weightage: 30% of the total marks
- End-semester exam
  - Weightage: 30% of the total marks
- Regular attendance: 10% of the total marks

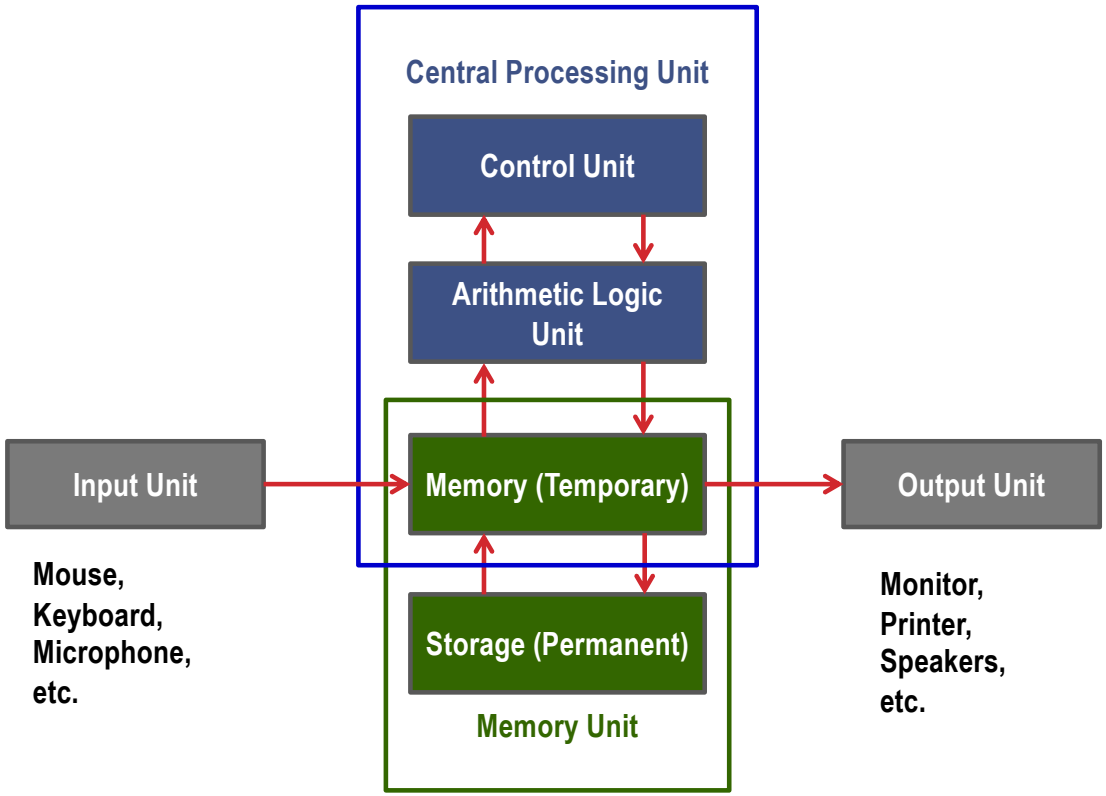
Lets start…

First things first: What is inside a computer?

Basic Components in a Computer



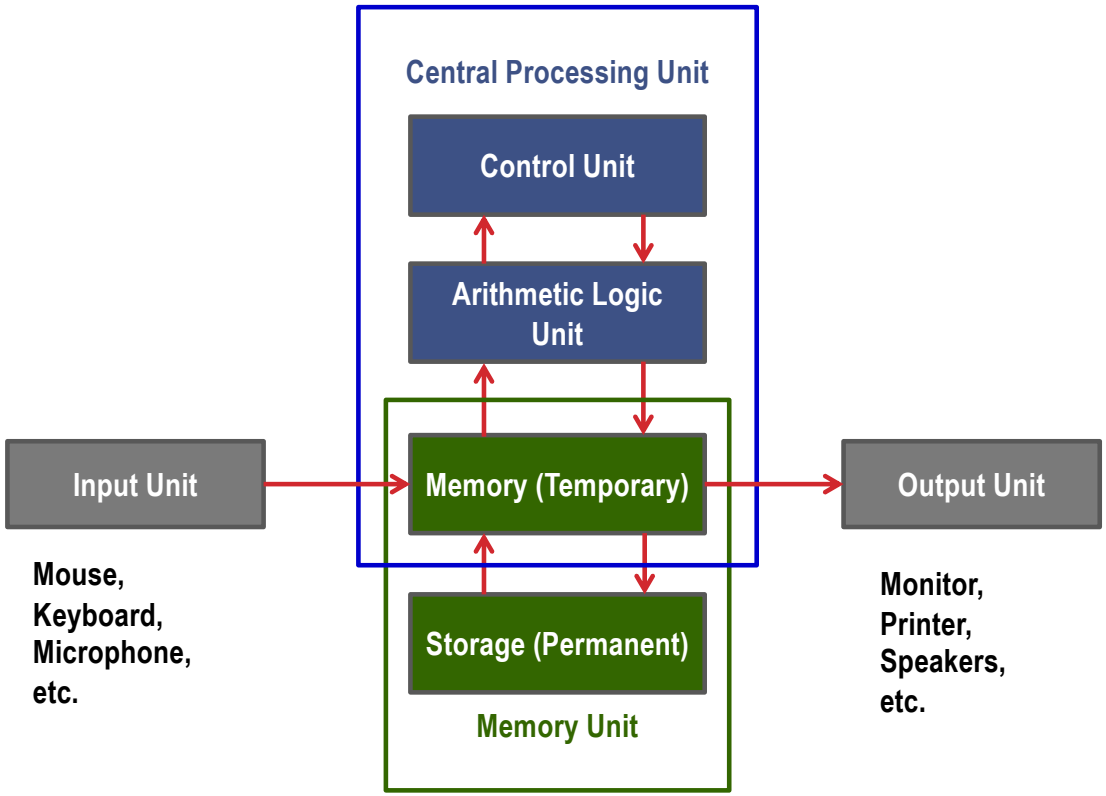
Basic Components in a Computer



CPU

- All computations take place here in order for the computer to perform a designated task.
- Has circuitry to carry out arithmetic and logic operations, take decisions, etc.
- CPU retrieves instructions from the memory (fetch), interprets (decode) them, and performs the requested operation (execute).

Basic Components in a Computer



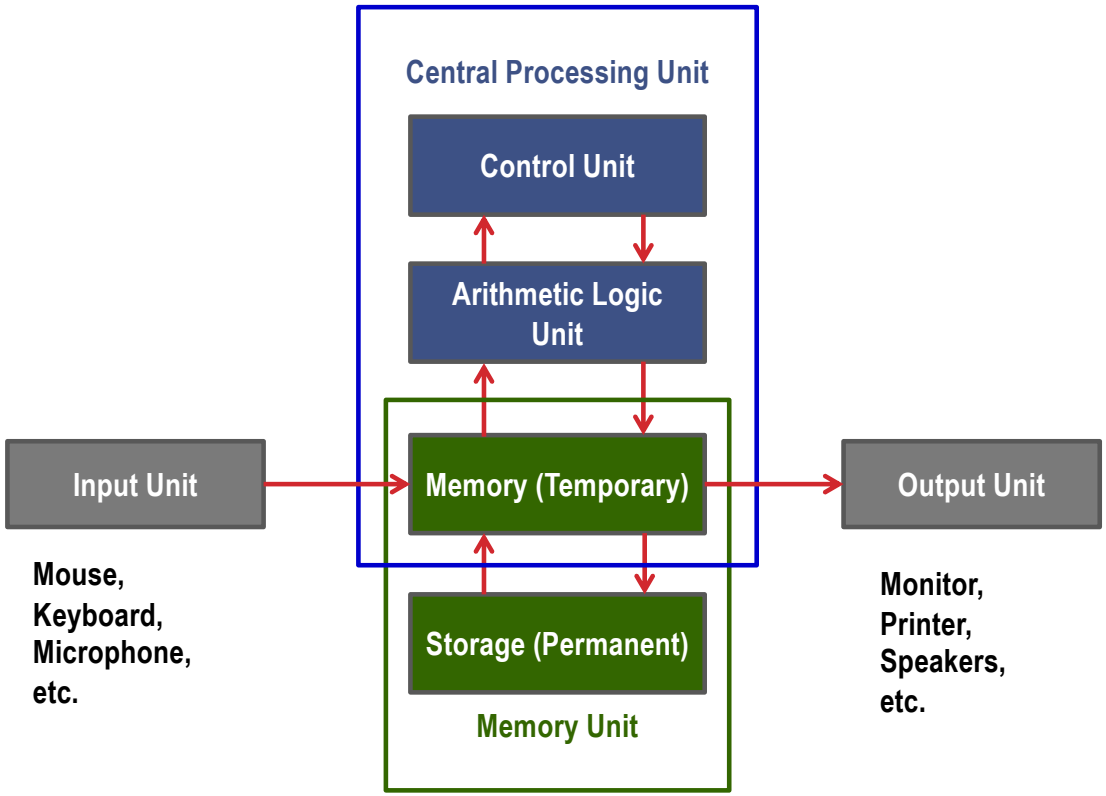
Memory unit

- Uses semiconductor technology
- Memory sizes in the range of 4 to 16 Gigabytes are typical today.

Some measures:

- 1 K (kilo) = 2^{10} (= 1024)
- 1 M (mega) = 2^{20} (= one million approx.)
- 1 G (giga) = 2^{30} (= one billion approx.)
- 1 T (tera) = 2^{40} (= one trillion approx.)

Basic Components in a Computer



Input Devices
Keyboard, Mouse, Scanner

Output Devices
Screen / Monitor, Printer

Storage Peripherals
Magnetic Disks: hard disk
Optical Disks: CD, DVD
Flash Memory
Solid State Drive

So why do we need to program a computer?

Super powers of computer

- This machine can do some tasks blazingly fast, tirelessly
  - Thousands of complex mathematical operations in a second

- But…
  - You need to speak its language
  - Tell computers exactly what operations to do, step by step
  - Programming…

Programming and Software

- A computer needs to be programmed to do tasks
- Programming - process of writing instructions in a language that can be understood by the computer so that a desired task can be performed by it
- Program: sequence of instructions to do a task, computer processes (executes) the instructions sequentially
- Software: programs for doing tasks on computers

Software

- **Software:** programs for doing tasks on computers
- Many software are already present in a computer that we buy
  - Applications: Editors, browsers, games, …
  - Operating System: makes the computer easy to use

Popular Operating Systems:

- **Windows:** single-user multitasking
- **Unix:** multi-user multitasking
- **Linux:** a free version of Unix

The laboratory classes will be based on Linux

These programs are written by others and provided to us when we buy a computer.
In this course, we will learn to write our own programs

Writing our own programs

- Programming - process of writing instructions in a language that can be understood by the computer so that a desired task can be performed by it

- What language does the computer understand?
  - Actually, understand only 1 and 0 (on and off); called "machine language"
  - We will write programs in a high-level programming language, which we will then translate to a form that the computer understands (executable code)

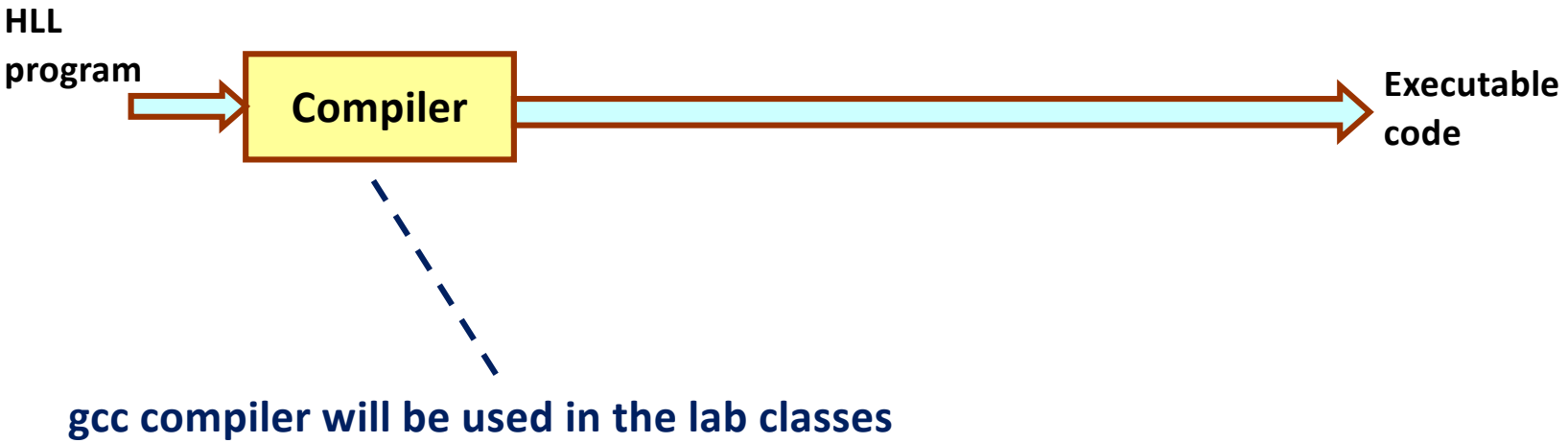
Okay… so how to write programs and make computer obey my instructions?

Three steps in writing programs

Step 1: Write the program in a high-level language (HLL - in your case, C)

Step 2: Compile the program using a C compiler

Step 3: Run / execute the program (ask the computer to execute it)

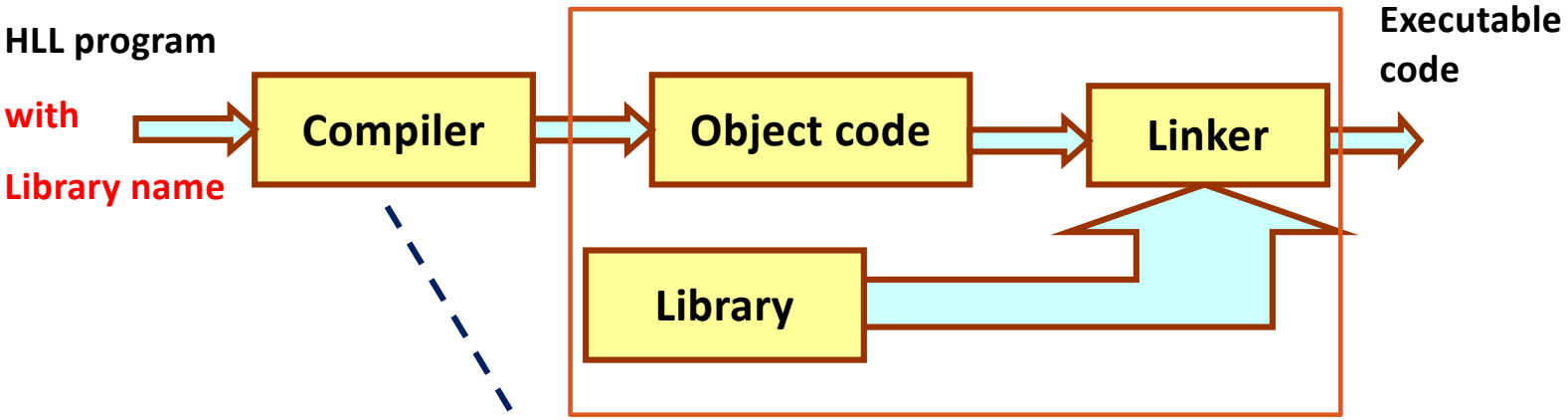


But, you can use instructions written by others: Libraries

Libraries: Pre-written instructions for complicated tasks

You can include these library names in your program and use their functionalities

Don't need to write those instructions yourself



gcc compiler will be used in the lab classes

This executable code is in binary
Computer understands only numbers, in binary (0 and 1)

Binary Representation

The decimal number system we use is base-10

- 10 digits, from 0 to 9, Positional weights 10^0, 10^1, 10^2,…from right to left for integers
- Example: $723 = 3x10^0 + 2x10^1 + 7x10^2$

Numbers are represented inside computers in the base-2 system (Binary Numbers)

- Only two symbols/digits 0 and 1
- Positional weights of digits: 2^0, 2^1, 2^2,…from right to left for integers
- Example: 1101 in binary $= (1 \times 2^0) + (0 \times 2^1) + (1 \times 2^2) + (1 \times 2^3) = 13$ in decimal

Binary Numbers

Dec	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

Binary Numbers

Dec	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000

Binary to Decimal Conversion

$101011 \rightarrow 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 = 43$

$(101011)_2 = (43)_{10}$

$111001 \rightarrow 1x2^5 + 1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 57$

$(111001)_2 = (57)_{10}$

$10100 \rightarrow 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0 = 20$

$(10100)_2 = (20)_{10}$

Bits and Bytes

- Bit – a single 1 or 0
- Byte – 8 consecutive bits
  - 2 bytes = 16 bits
  - 4 bytes = 32 bits
- Maximum integer that can represented
  - in 1 byte = 255 (=11111111)
  - In 4 bytes = 4294967295 (= 32 1's)
- Number of integers that can be represented in 1 byte = 256 (the integers 0, 1, 2, 3,….255)

Problem Solving
(the thought process)

Problem solving: typical flow

Step 1:

- Clearly specify the problem to be solved.

Step 2:

- Draw flowchart or write algorithm (represent the sequence of operations needed).

Step 3:

- Convert flowchart (algorithm) into program code.

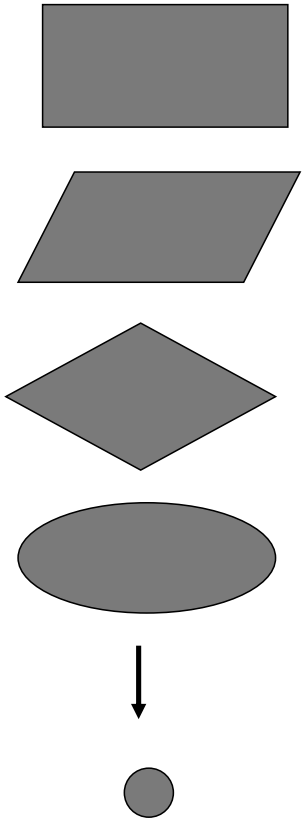
Step 4:

- Compile the program to get executable code.

Step 5:

- Execute the program on a computer.

Flowchart: represent the sequence of operations as a diagram



Computation

Input / Output

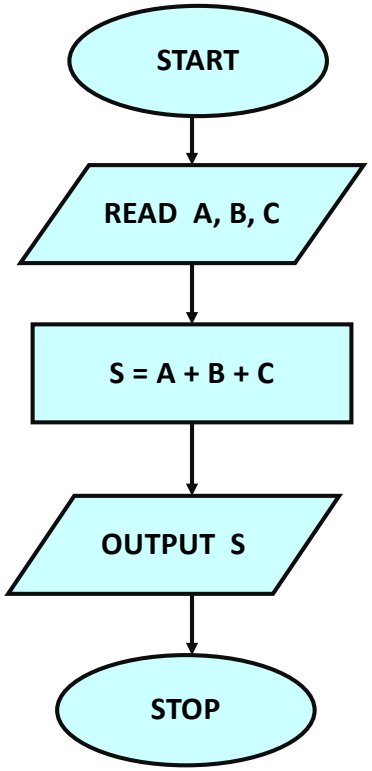
Decision Box

Start / Stop

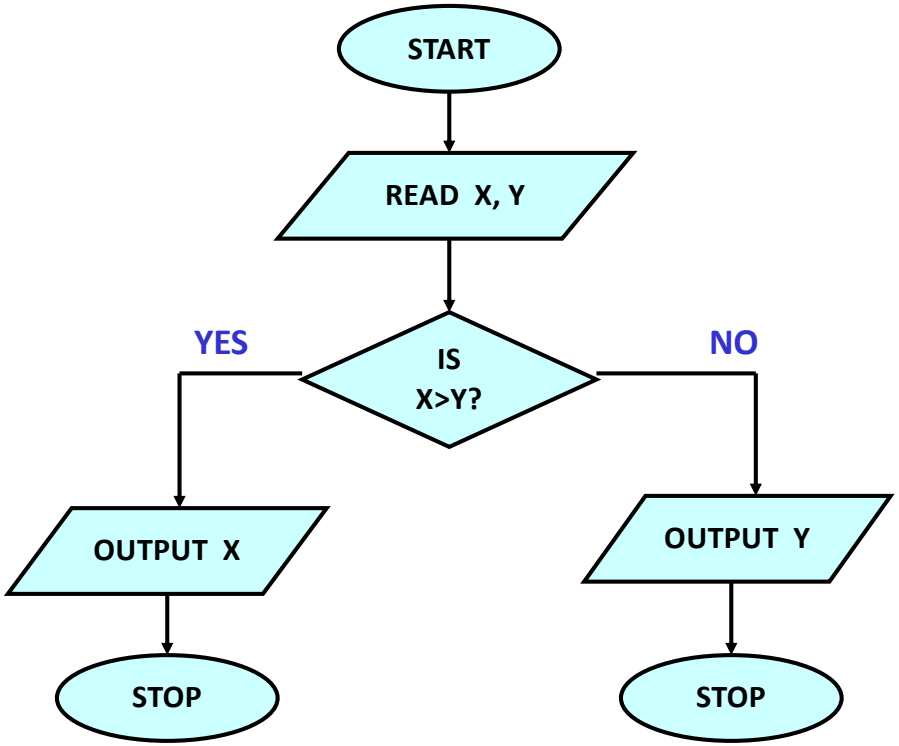
Flow of control

Connector

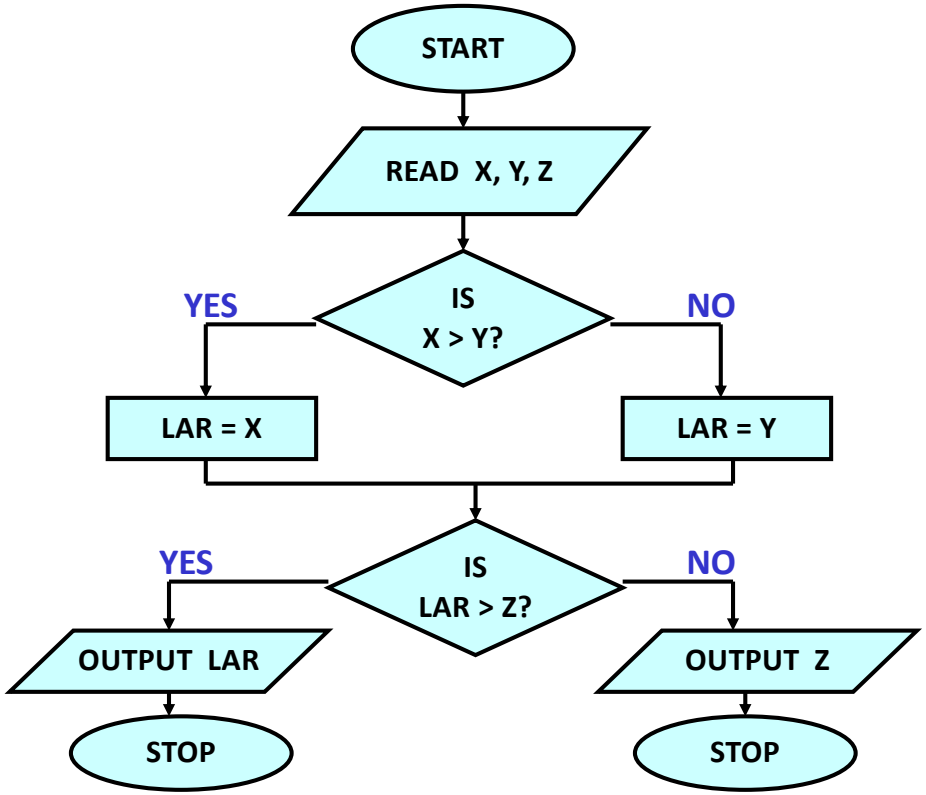
Flowchart: Example 1: adding three numbers



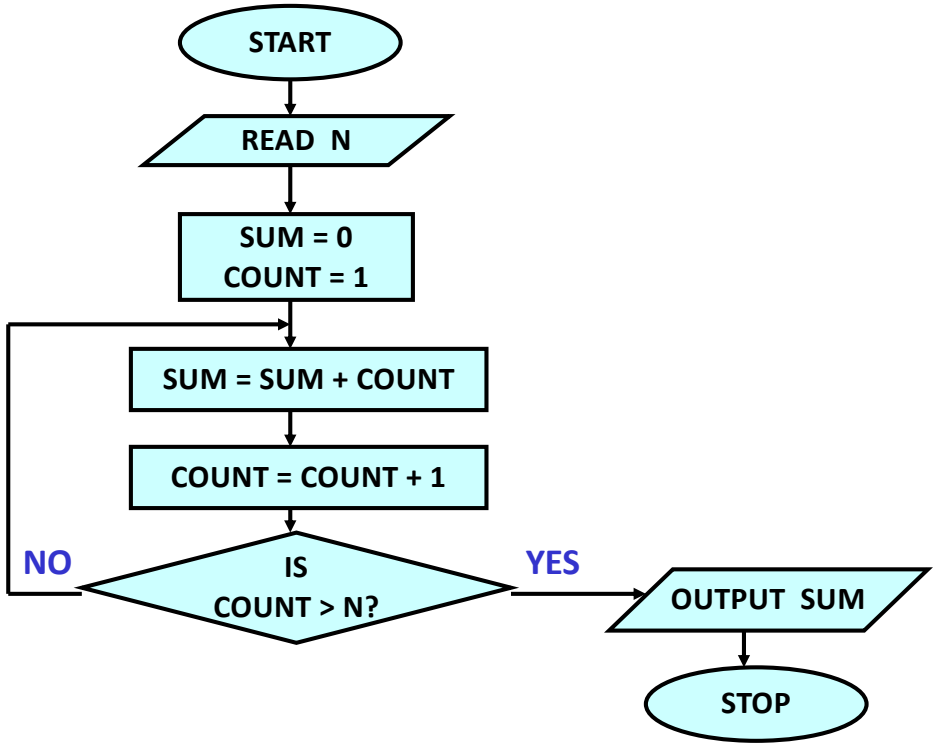
Flowchart: Example 2: find larger of two numbers



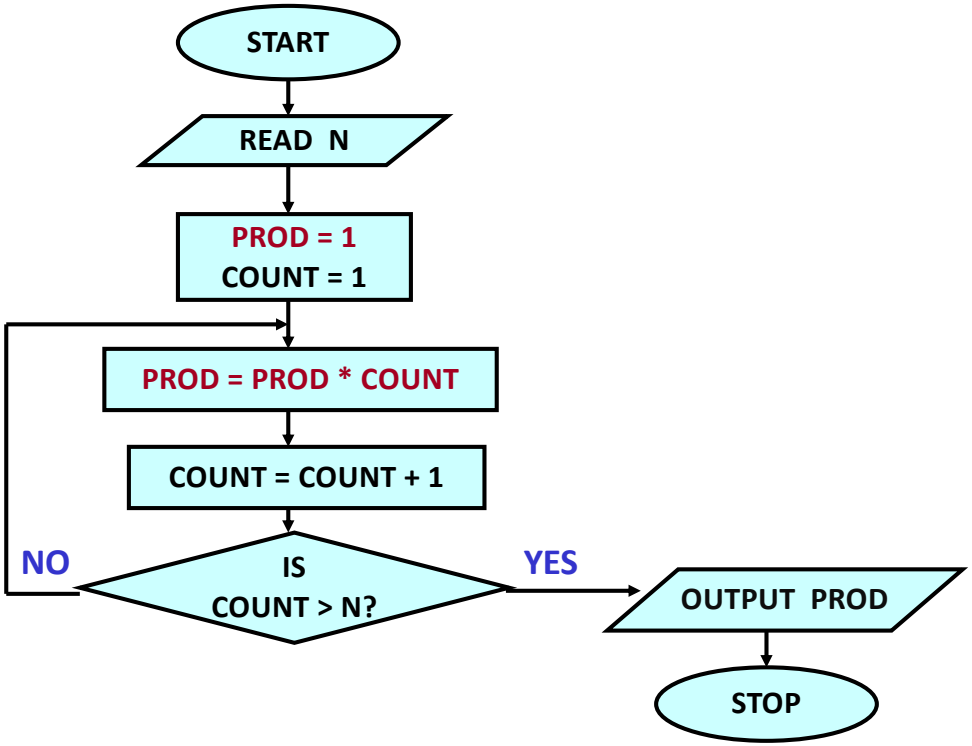
Flowchart: Example 3: find largest of three numbers



Flowchart: Example 4: sum first N integers



Flowchart: Example 5: compute factorial of a number



Fundamentals of C

First C program – print on screen

```
#include <stdio.h>
int main()
{
    printf ("Hello, World! \n") ;
    return 0;
}
```

Output

```
Hello, World!
```

A Simple C program

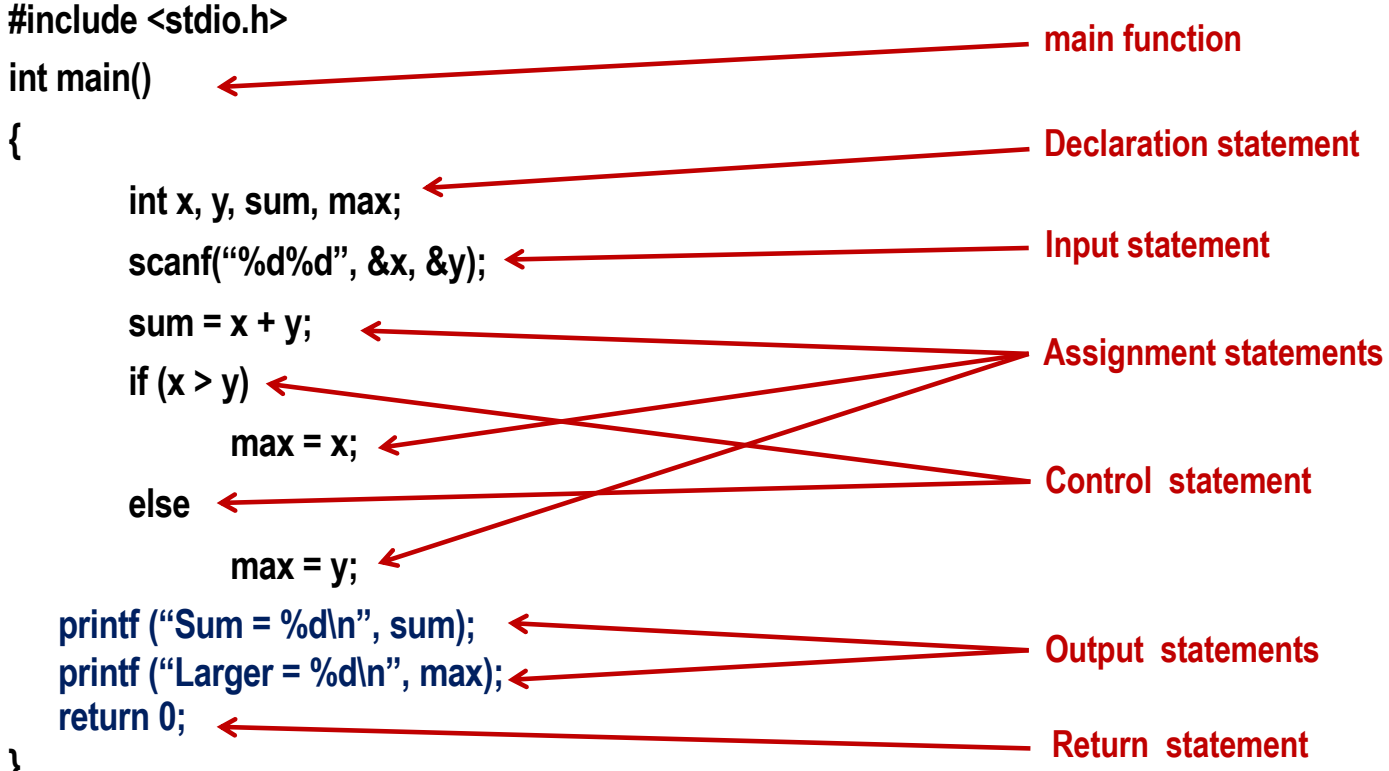
```
#include <stdio.h>
int main()
{
        int x, y, sum, max;

        scanf("%d%d", &x, &y);

        sum = x + y;
        if (x > y) max = x;
                else  max = y;
        printf ("Sum = %d\n",  sum);
        printf ("Larger = %d\n",  max);
        return 0;
}
```

When you run the program

Output after you type 15 and 20

```
15 20
Sum = 35
Larger = 20
```

```
#include <stdio.h>
int main()
{
        int x, y, sum, max;
        scanf("%d%d", &x, &y);
        sum = x + y;
        if (x > y)
                max = x;
        else
                max = y;
    printf ("Sum = %d\n", sum);
    printf ("Larger = %d\n", max);
    return 0;
}
```



Structure of a C program

- A collection of functions (we will see what they are later)
- Exactly one special function named main( ) must be present. Program always starts from there.
  - Until we study functions in detail, this is the only function your programs will have for now
- Each function has different types of statements
  - variable declarations, assignment, condition check, looping etc.
- Statements are executed one by one in top-down order (though some special instructions differ from this top-down order)

Writing a C program

- You will have to understand what different statements do to decide which you should use in what order to solve your problem
- There is a fixed format (“syntax") for writing each statement and other things. Need to remember the syntax.
  - Syntax – grammar of a programming language
  - Compiler will give error if your typed program does not match required C syntax
- There are other rules to follow

Things you will see in a C program (we will look at all these one by one)

- Variables
- Constants
- Expressions (Arithmetic, Logical, Assignment)
- Statements (Declaration, Assignment, Control (Conditional/Branching, Looping))
- Arrays
- Functions
- Structures
- Pointers

The C Character Set

The C language alphabet

- Uppercase letters ‘A’ to ‘Z’
- Lowercase letters ‘a’ to ‘z’
- Digits ‘0’ to ‘9’
- Certain special characters:

!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
|	;	:	‘	“	{	}	,
.	<	>	/	?			

whitespace characters (space, tab, …)

A C program should not contain anything else

Variables

- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program

Variables are stored in the computer's memory

Memory is a list of consecutive storage locations, each having a unique address

Four aspects of a variable:

- The value stored in the variable
- The type of the variable – decides what type of value can be stored (integer, real number, etc.)
- The variable name is used to refer to the value of the variable
- A variable is stored at a particular location of the memory, called its address

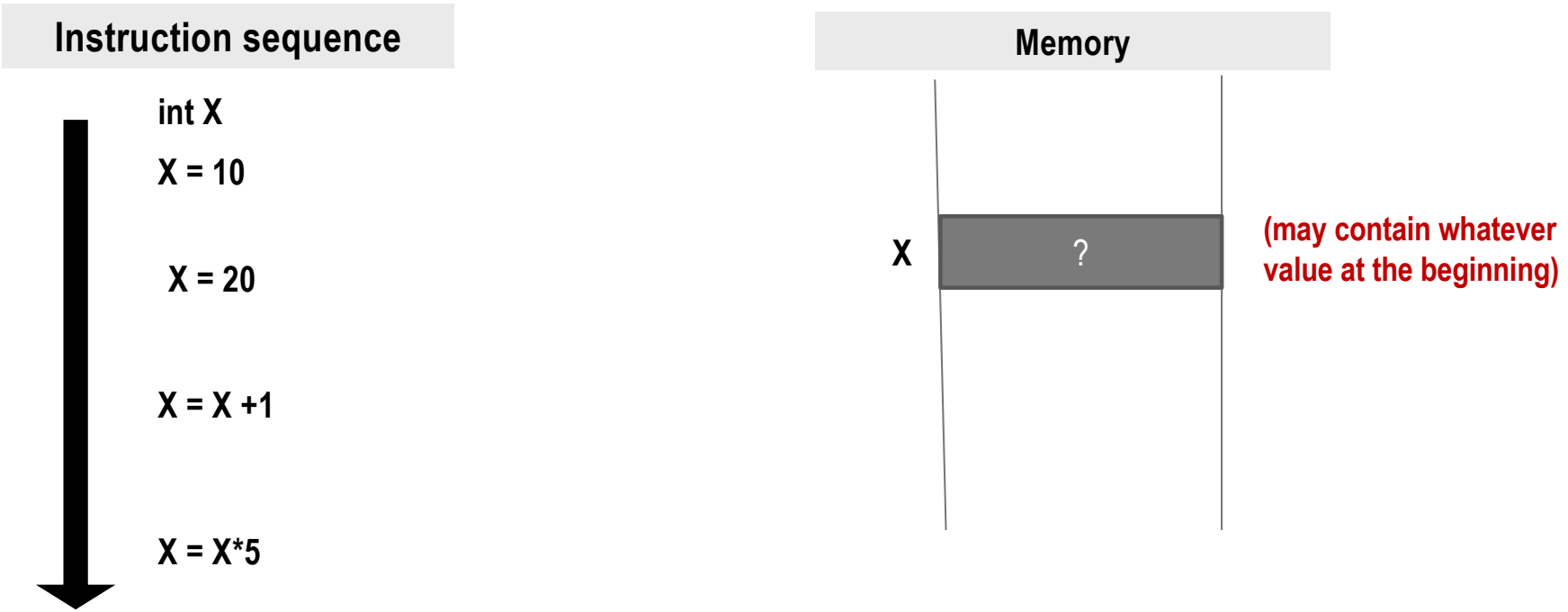
Example

```
#include <stdio.h>
int main( )
{
        int x;
        int y;

        x=1;
        y=3;

        printf("x = %d, y= %d\n", x, y);
        return 0;
}
```

The = sign in a C program does NOT denote equality

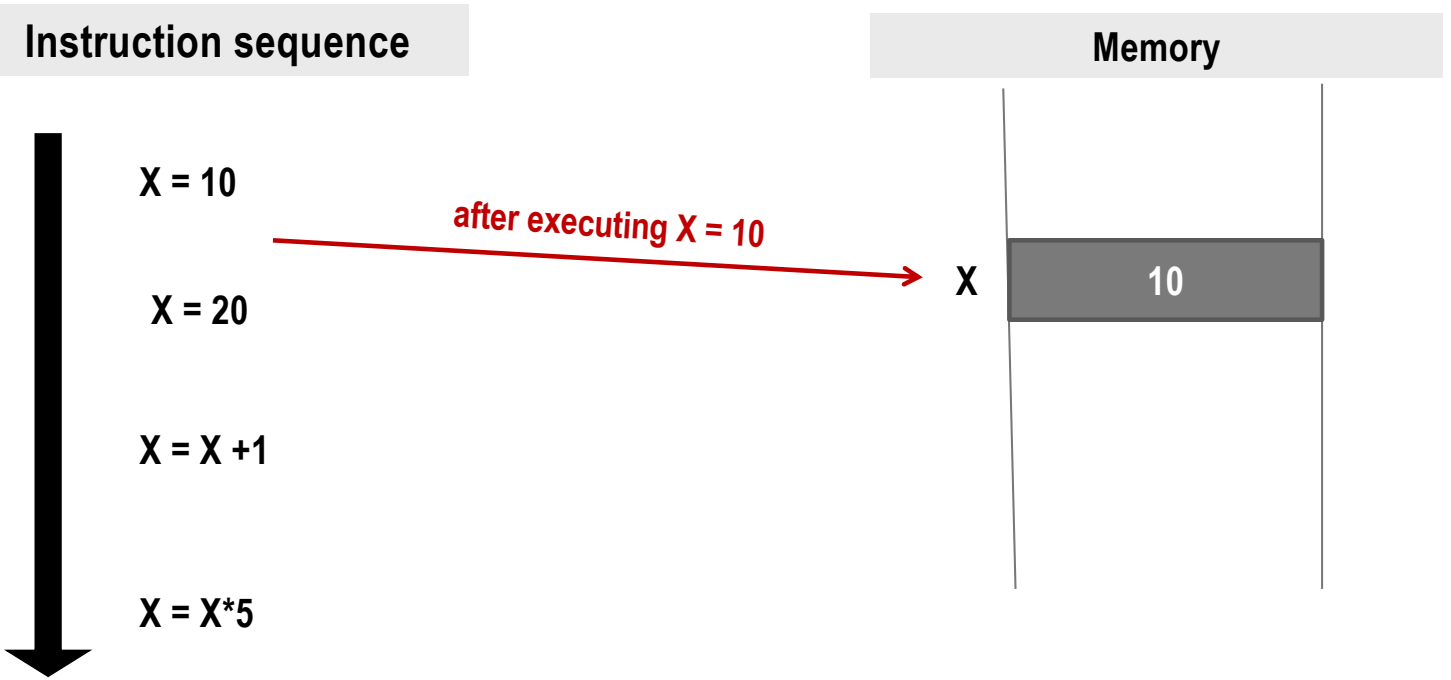
We are actually assigning a value to a variable, i.e., storing a particular value into the variable

Variables in Memory



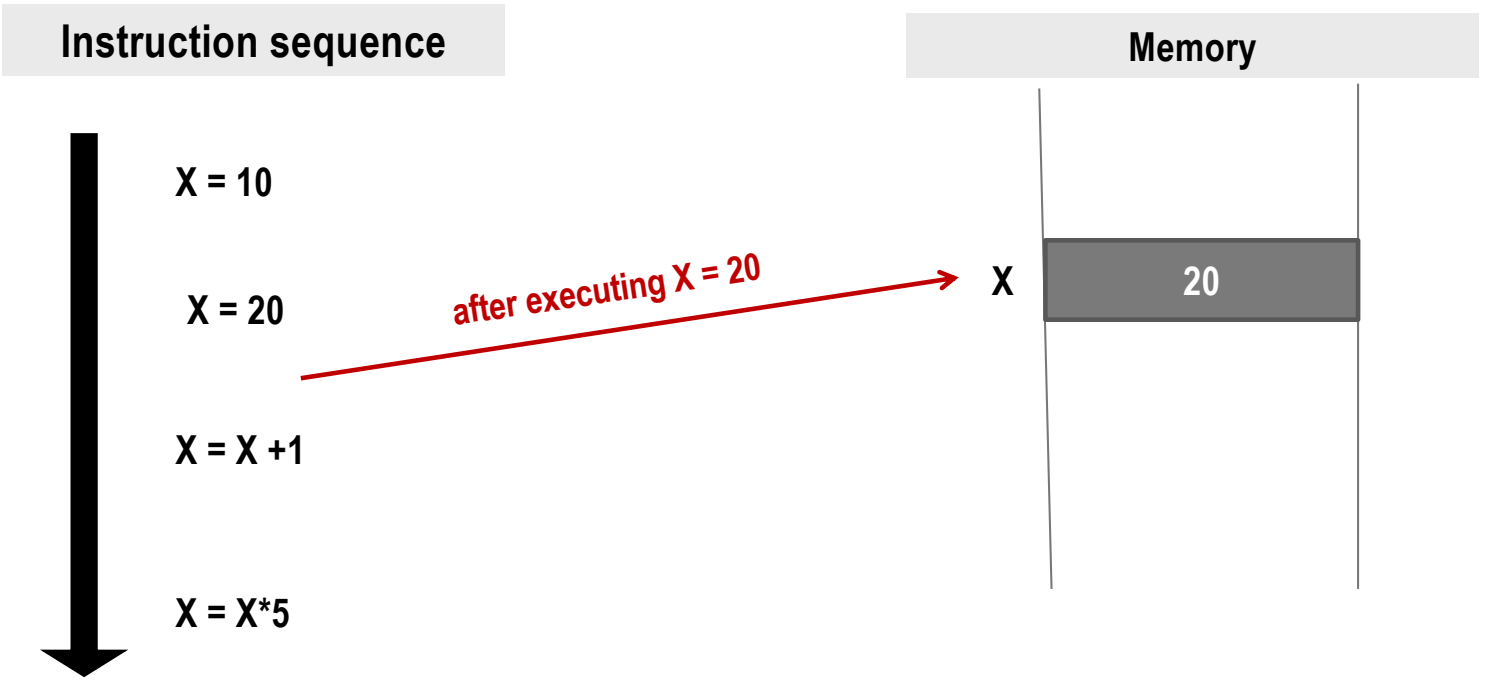
A variable can have only one value assigned to it at any given time during the execution of the program

Variables in Memory



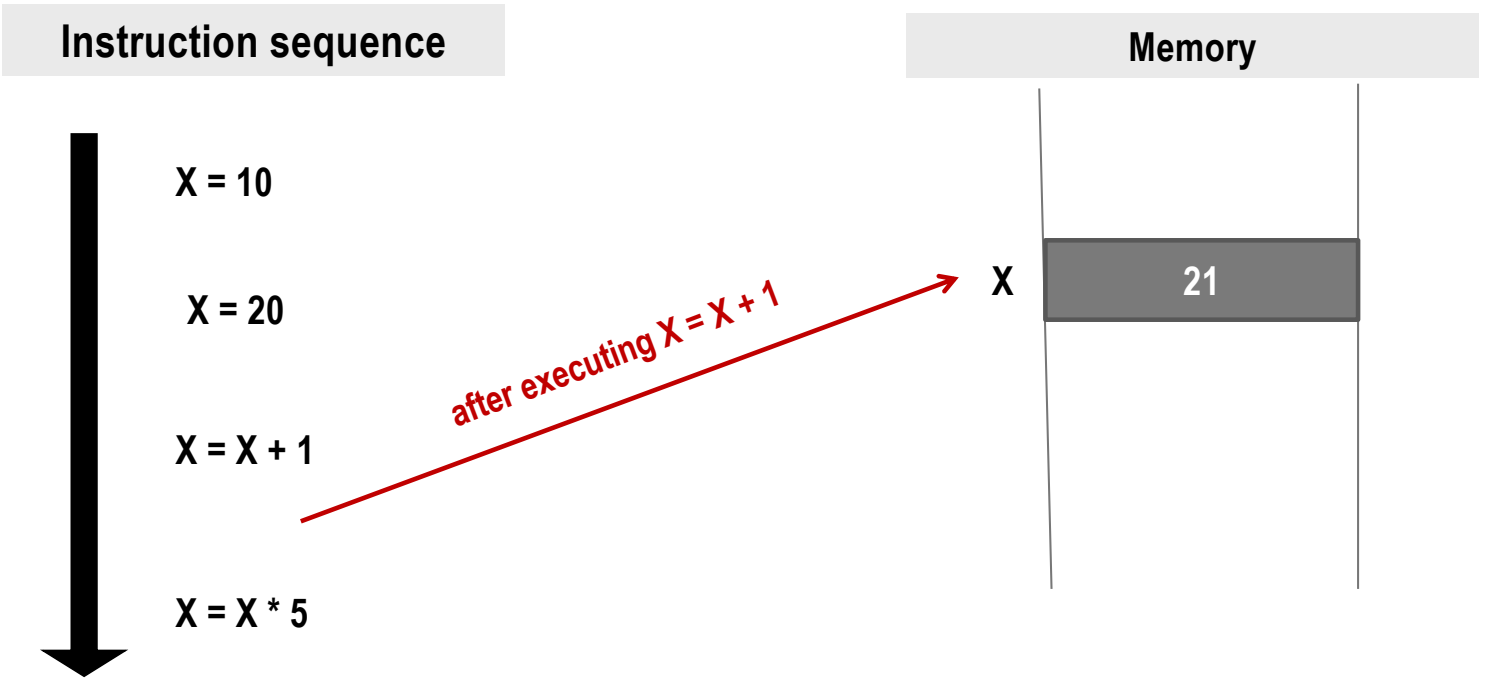
A variable can have only one value assigned to it at any given time during the execution of the program

Variables in Memory



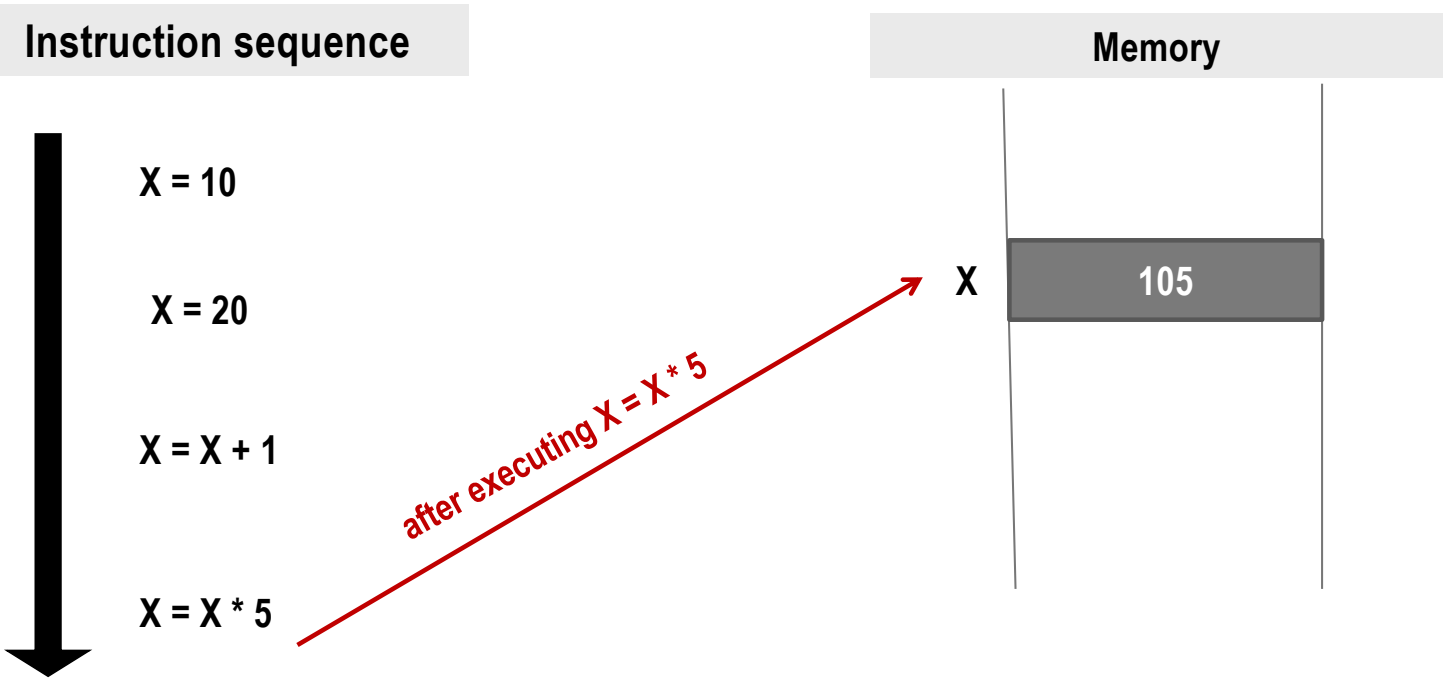
A variable can have only one value assigned to it at any given time during the execution of the program

Variables in Memory



A variable can have only one value assigned to it at any given time during the execution of the program

Variables in Memory



A variable can have only one value assigned to it at any given time during the execution of the program

Variables in Memory

Instruction sequence

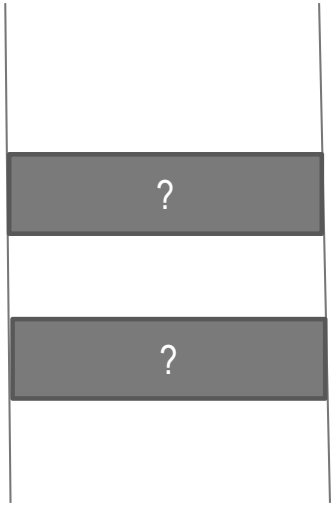
X = 20

Y = 15

X = Y + 3

Y = X / 6

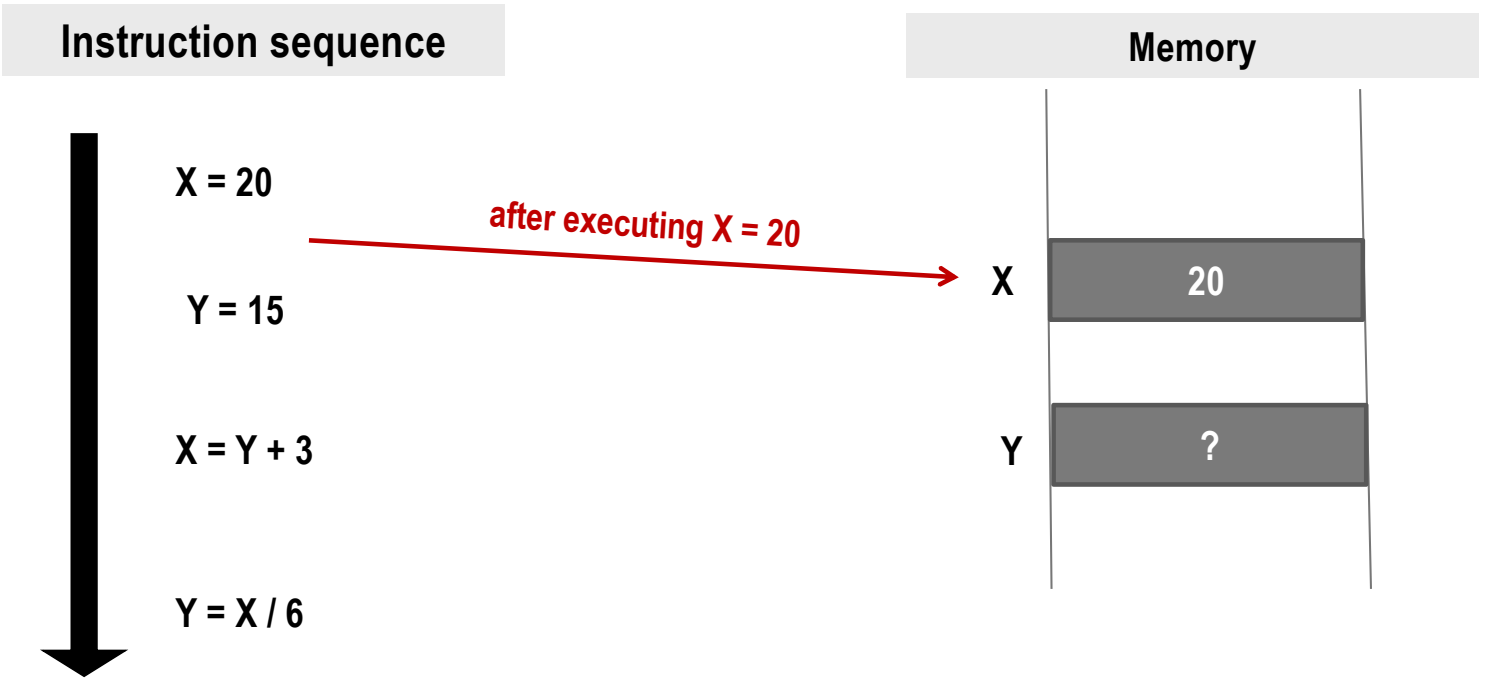
Memory



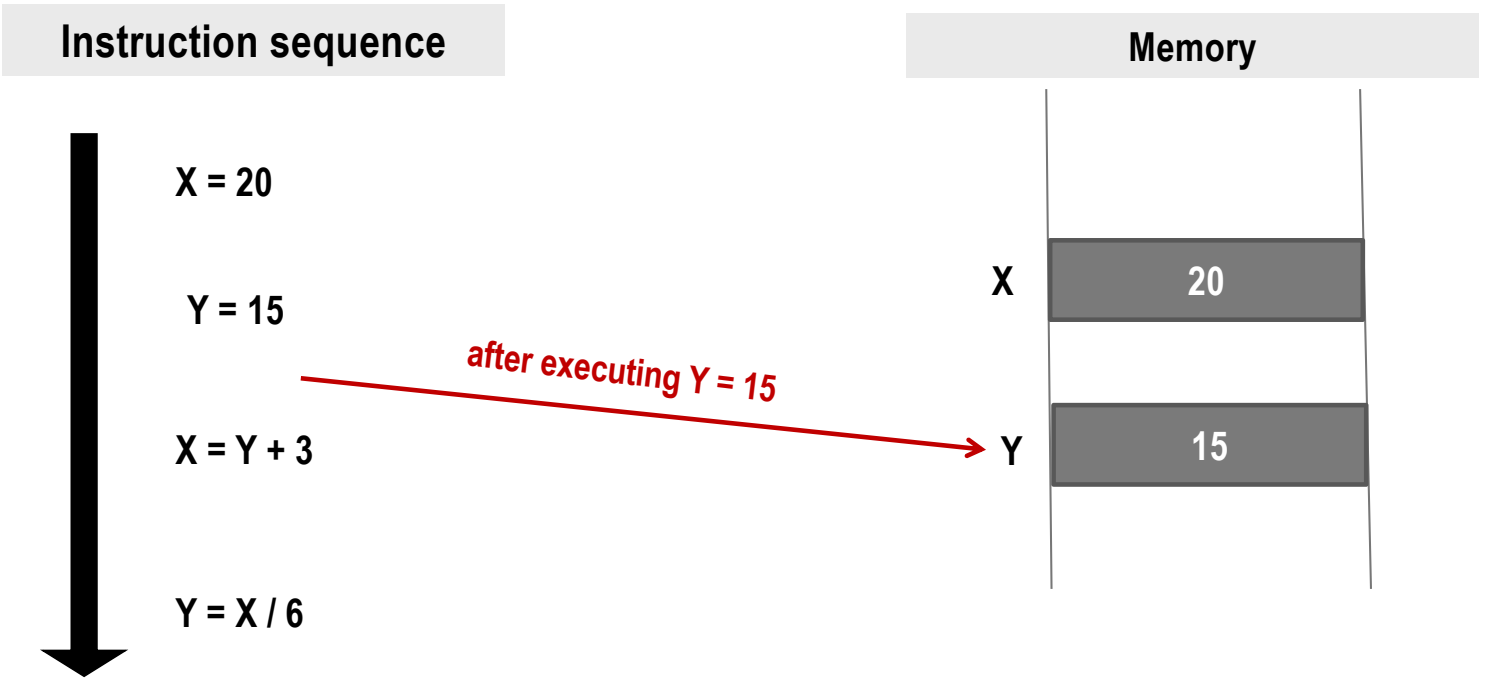
(may contain whatever value at the beginning)

(may contain whatever value at the beginning)

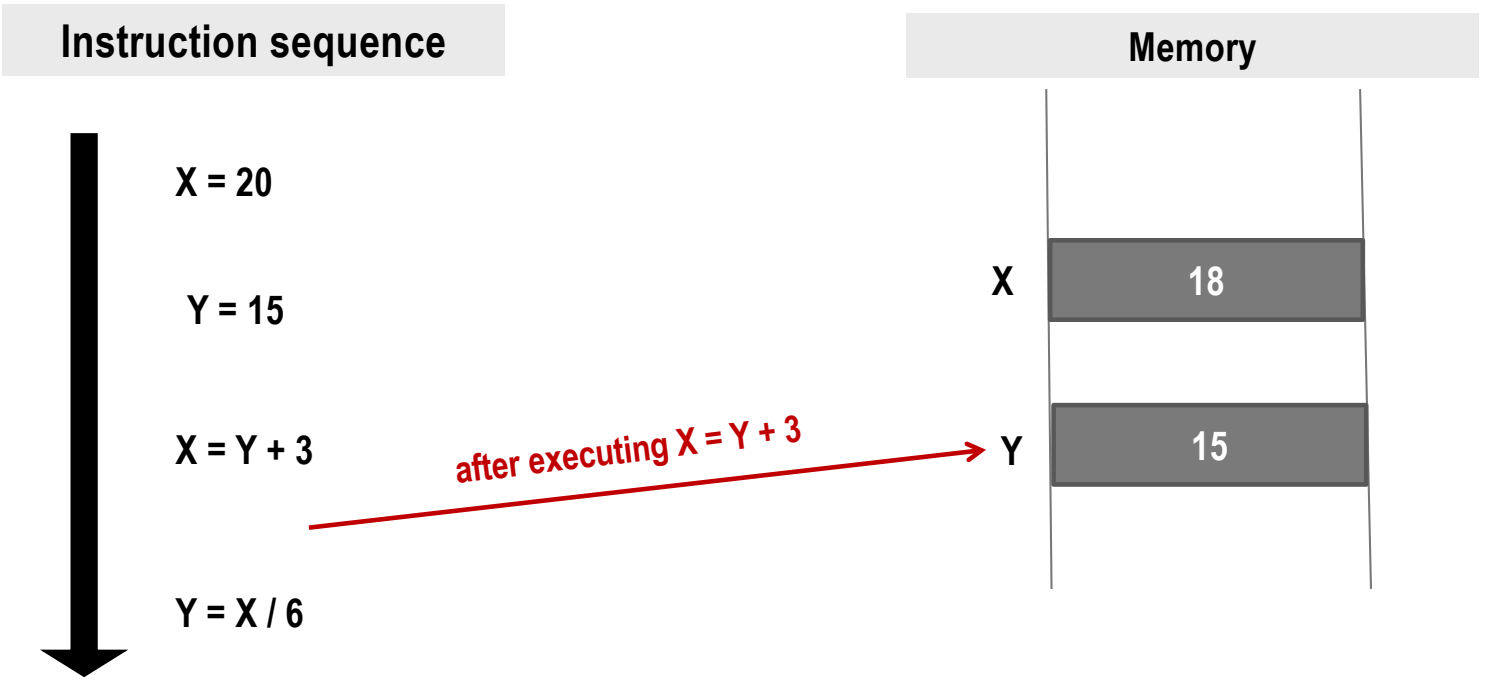
Variables in Memory



Variables in Memory

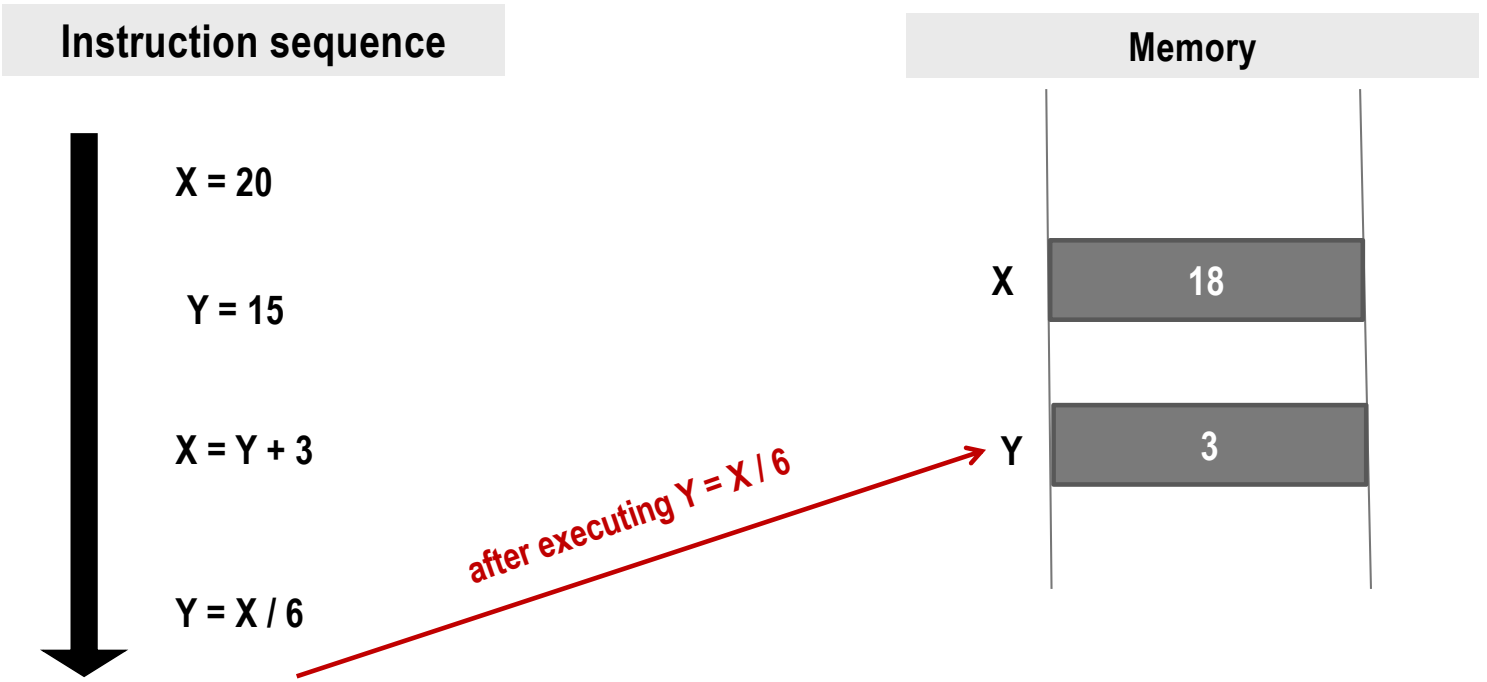


Variables in Memory



Note: Y does not change, only X changes after execution of this instruction

Variables in Memory



Data Types

- Each variable has a type, indicates what type of values the variable can hold
- Four common data types in C (there are others)
  - int - can store integers (usually 4 bytes)
  - float - can store single-precision floating point numbers (usually 4 bytes)
  - double - can store double-precision floating point numbers (usually 8 bytes)
  - char - can store a character (1 byte)

Rules and good practices

- First rule of variable use: Must declare a variable (specify its type and name) before using it anywhere in your program

- All variable declarations should ideally be at the beginning of the main( ) or other functions
  - There are exceptions, we will see later

- A value can also be assigned to a variable at the time the variable is declared.

```
int   speed = 30;
char  flag = 'y';
```

Initialization of a variable

Variable Names

- Sequence of letters and digits
- First character must be a letter or '_'
- No special characters other than '_'
- No blank in between
- Names are case-sensitive (max and Max are two different names)

- Examples of valid names:

  i rank1 MAX max Min class_rank

- Examples of invalid names:

  a's fact rec 2sqroot class,rank

More Valid and Invalid Identifiers

Valid identifiers

- X
- abc
- simple_interest
- a123
- LIST
- stud_name
- Empl_1
- Empl_2
- _avg_empl_salary

Invalid identifiers

- 10abc
- my-name
- “hello”
- simple interest
- (area)
- %rate

C Keywords

- Specific terms used by the C language having specific meaning, cannot be used as variable names
- Examples:

  int, float, char, double, main, if else, for, while, do, struct, union, typedef, enum, void, return, signed, unsigned, case, break, sizeof,….

  - There are others, see any good textbook…

Example 1

```
#include <stdio.h>
int main()
{
    int x, y, sum;

    scanf("%d%d", &x, &y);
    sum = x + y;

    printf( "%d plus %d is %d\n", x, y, sum );

    return 0;
}
```

Three integer type variables declared

Values assigned

Example 2

```
#include <stdio.h>
int main()
{
    float x, y;                 /* Two floating point variables declared */
    int d1, d2 = 10;            /* Here integer variable d2 is initialized to 10, can be changed later */


        scanf("%f%f%d",&x, &y, &d1);
        printf( "%f plus %f is %f\n", x, y, x + y);
        printf( "%d minus %d is %d\n", d1, d2, d1 – d2);

    return 0;

}
```

Read-only Variables

- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the const keyword in front of the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed
  - Prevents accidental change of the value

Correct

```
int main( ) {
   const int LIMIT = 10;
   int n;
   scanf("%d", &n);
   if (n > LIMIT) printf("Out of limit");
   return 0;
}
```

Incorrect: Limit changed

```
int main() {
   const int Limit = 10;
   int n;
   scanf("%d", &n);
   Limit = Limit + n;
   printf("New  limit is %d", Limit);
   return 0;
}
```

Constants

- Integer constants
  - Consists of a sequence of digits, with possibly a plus or a minus sign before it
  - Embedded spaces, commas and non-digit characters are not permitted between digits
  - Examples: 10, 39994, -765

- Floating point constants

- Two different notations:
  - Decimal notation: 25.0, 0.0034, .84, -2.234
  - Exponential (scientific) notation
    - 3.45e23, 0.123e-12, 123e2

e means “10 to the power of”

Constants (Contd.)

- Character constants
  - Contains a single character enclosed within a pair of single quote marks.
  - Examples :: ‘2’, ‘+’, ‘Z’
  - Note: ‘2’ (character constant) is different from 2 (integer constant)

- Some special backslash characters – these are also character constants
  - ‘\n’ new line
  - ‘\t’ horizontal tab
  - ‘\’’ single quote
  - ‘\”’ double quote
  - ‘\\’ backslash
  - ‘\0’ null

Typical Size of Data Types

char – 1 byte

int – 4 bytes

float – 4 bytes

double – 8 bytes

- "Typical", because some of them vary depending on machine/OS type
- Therefore never use the values (1, 4, 8) directly, use the sizeof( ) operator given
  - sizeof(char) will give 1, sizeof(int) will give 4 and so on … can vary as per your PC/Laptop

Input: scanf function

- Performs input from keyboard
- It requires a format string and a list of variables into which the value received from the keyboard will be stored
- format string = individual groups of characters (usually '%' sign, followed by a conversion character), with one character group for each variable in the list

```
int a, b;
float c;
scanf( "%d%d%f", &a, &b, &c);
```

Variable list (note the & before each variable name)

Format string

Commonly used conversion characters

c	for char type variable
d	for int type variable
f	for float type variable
lf	for double type variable

Examples

scanf ("%d", &size) ;

- Reads one integer from keyboard into an int type variable named size

scanf ("%c", &nextchar) ;

- Reads one character from keyboard into a char type variable named nextchar

scanf ("%f", &length) ;

- Reads one floating point (real) number from keyboard into a float type variable named length

scanf ("%d%d", &a, &b);

- Reads two integers from keyboard, the first one in an int type variable named a and the second one in an int type variable named b

Important

- scanf( ) will wait for you to type the input from the keyboard
- You must type the same number of inputs as the number of %'s in the format string
- Example: if you have scanf("%d%d",…), then you must type two integers (in same line or different lines), otherwise scanf( ) will just wait and the next statement will not be executed

Reading a single character

- A single character can be read using scanf with %c
- It can also be read using the getchar() function

```
char c;
c = getchar();
```

- Program waits at the getchar() line until a character is typed, and then reads it and stores it in c

Output: printf function

- Performs output to the standard output device (typically defined to be the screen)
- It requires a format string in which we can specify:
  - The text to be printed out
  - Specifications on how to print the values

    printf ("The number is %d\n", num);
  - The format specification %d causes the value listed after the format string to be embedded in the output as a decimal number in place of %d
  - Output will appear as: The number is 125

General syntax of printf( )

printf (format string, arg1, arg2, …, argn);

- format string refers to a string containing formatting information and data types of the arguments to be output
- the arguments arg1, arg2, … represent list of variables/expressions whose values are to be printed
- The conversion characters are the same as in scanf

- Examples:

```
printf (“Average of %d and %d is %f”, a, b, avg);
printf (“Hello \nGood \nMorning \n”);
printf(“%3d %3d %5d”, a, b, a*b+2);
printf(“%7.2f  %5.1f”, x, y);
```

- Many more options are available for both printf and scanf – read from the book

More Examples

(Explain the outputs to test if you understood format strings etc.)

More print

```
#include <stdio.h>
int main()
{
    printf ("Hello, World! ") ;
    printf ("Hello \n World! \n") ;
    return 0;
}
```

Output

```
Hello, World! Hello
 World!
```

Some more print

```
#include <stdio.h>

int main()

{

    printf ("Hello, World! \n") ;

    printf ("Hello \n World! \n") ;

    printf ("Hell\no \t World! \n") ;

    return 0;

}
```

Output

```
Hello, World!
Hello
 World!
Hell
o        World!
```

Some more print

```
#include <stdio.h>
int main()
{
        float f1, f2;
        int x1, x2;
        printf("Enter values for f1 and f2: \n");
        scanf("%f%f", &f1, &f2);
        printf("Enter values for x1 and x2: \n");
        scanf("%d%d", &x1, &x2);
        printf("f1 = %f, f2 = %8.2f\n", f1, f2);
        printf("x1 = %d, x2 = %10d\n", x1, x2);
        return 0;
}
```

Output

```
Enter values for f1 and f2:
23.5  14.326
Enter values for x1 and x2:
54  7
f1 = 23.500000, f2 =     14.33
x1 = 54, x2 =          7
```

Can you explain why 14.326 got printed as 14.33?

Practice Problems

Write C programs to

1. Read two integers and two floating point numbers, each in a separate scanf() statement (so 4 scanf's) and print them with separate printf statements (4 printf's) with some nice message
2. Repeat 1, but now read all of them in a single scanf statement and print them in a single printf statement
3. Repeat 1 and 2 with other data types like double and char
4. Repeat 1 and 2, but now print all real numbers with only 3 digits after the decimal point
5. Read 4 integers in a single scanf statement, and print them (using a single printf statement) in separate lines such that the last digit of each integer is exactly 10 spaces away from the beginning of the line it is printed in (the 9 spaces before will be occupied by blanks or other digits of the integer). Remember that different integers can have different number of digits
6. Repeat 5, but now the first integer of each integer should be exactly 8 spaces away from the beginning of the line it is printed in.