CS10001:
Programming & Data Structures

Dept. of Computer Science & Engineering

Course Materials

- Slides available at http://cse.iitkgp.ac.in/pds/current
- More materials available at http://cse.iitkgp.ac.in/pds

Books:

1. **Programming with C**
   Byron Gottfried
2. **The C Programming Language**
   Brian W Kernighan, Dennis M Ritchie
3. **Data structures**
   S. Lipschutz, Schaum's Outline Series

About the Course

- Section 9, 10
  - Mon (10:00-10:55), Wed (8:00-9:55)
  - In Classroom NR-122 (Nalanda Complex)
  - Teacher: Prof. Rajib Mall (RM)
- Section 11, 12
  - Mon (10:00-10:55), Wed (8:00-9:55)
  - In Classroom NR-222 (Nalanda Complex)
  - Teacher: Prof. Arobinda Gupta (AG)
- Section 13, 14
  - Thurs (9:00-9:55), Fri (11:00-12:55)
  - In Classroom NR-122 (Nalanda Complex)
  - Teacher: Prof. Partha Pratim Das (PPD)
- Section 15, 16
  - Teacher: Prof. Shamik Sural (SSL)
  - Thurs (9:00-9:55), Fri (11:00-12:55)
  - In Classroom NR-222 (Nalanda Complex)

Evaluation

- Mid-semester 30%
- End-semester 50%
- Two class tests 20%

Attendance is Mandatory

- Important for understanding the course
- Note the Institute Policy:
  - "If a student does not have a minimum of 80% attendance in a subject, he/she can be de-registered from the subject at the discretion of the subject teacher"
- Leave Policy
  - Medical reasons (must be certified by the B.C. Roy Technology Hospital), or
  - Prior permission from Dean (Academic)

Important Dates

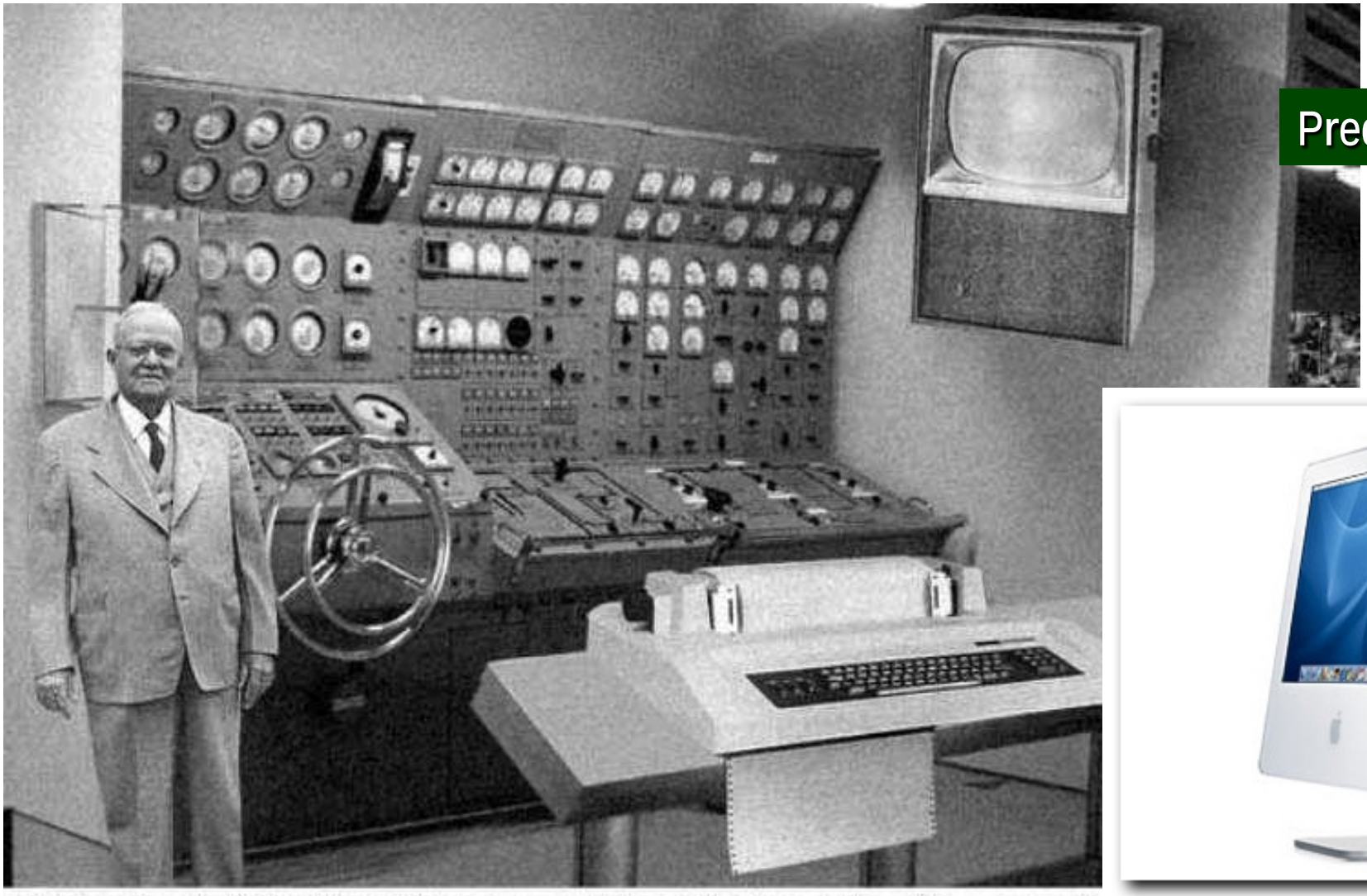
- Class Test 1: August 30, 2017 (19:00 – 20:00)
- Class Test 2: November 1, 2017 (19:00 – 20:00)

(Class test dates are tentative and may change. The exact dates will be announced in the class)

- Mid-semester: Sept 15 – Sept 25, 2017
- End-semester: Nov 20 – 28, 2017

Introduction

Home Computer@2004: *Predicted versus Real*

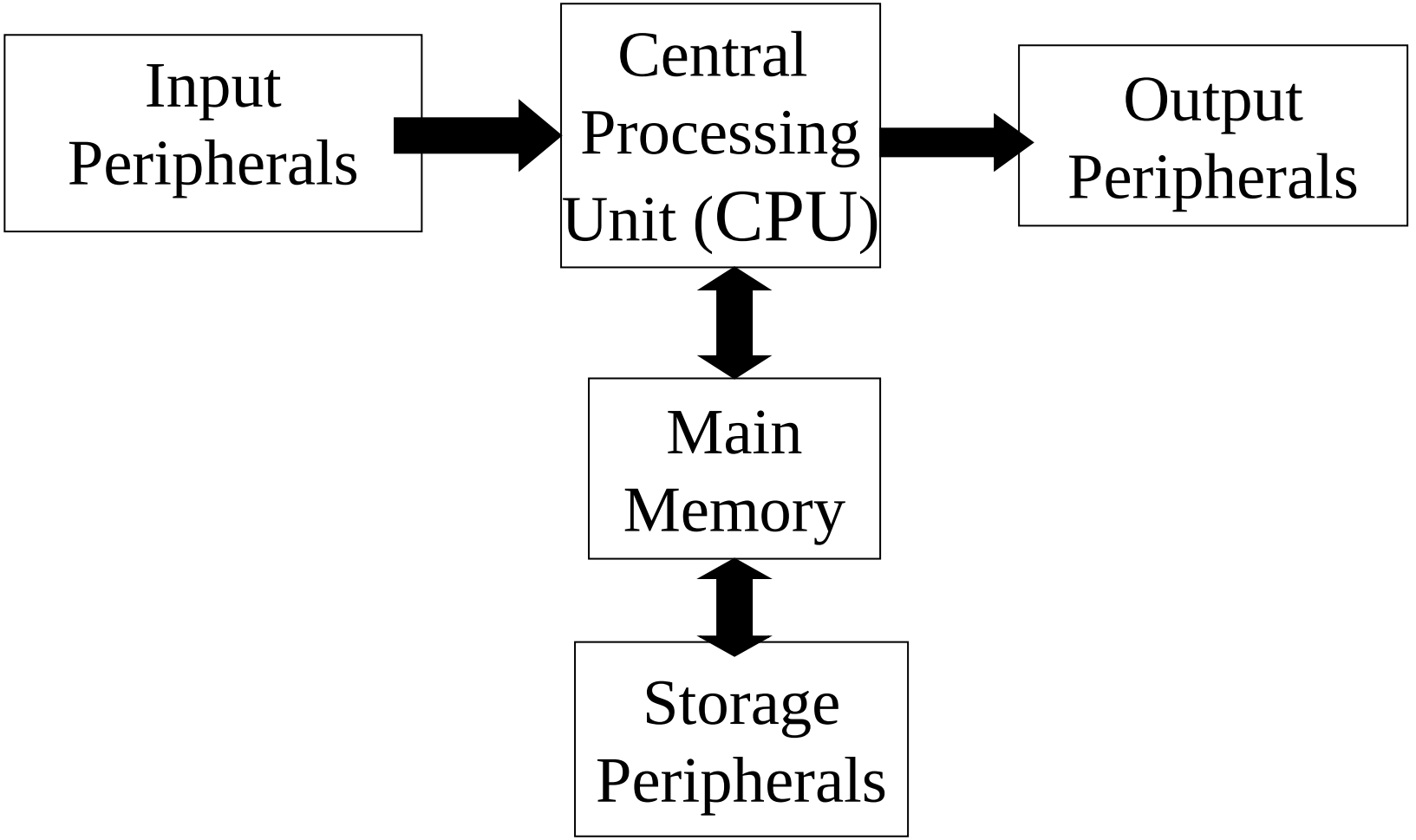


Predicted in 1954

Reality

Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

A Computer (Level 0 Version)

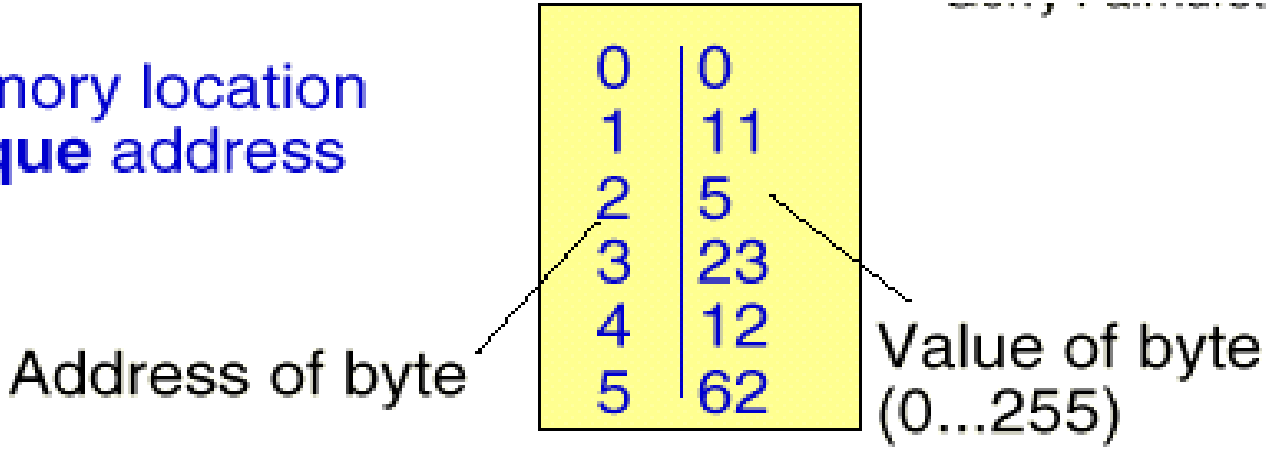


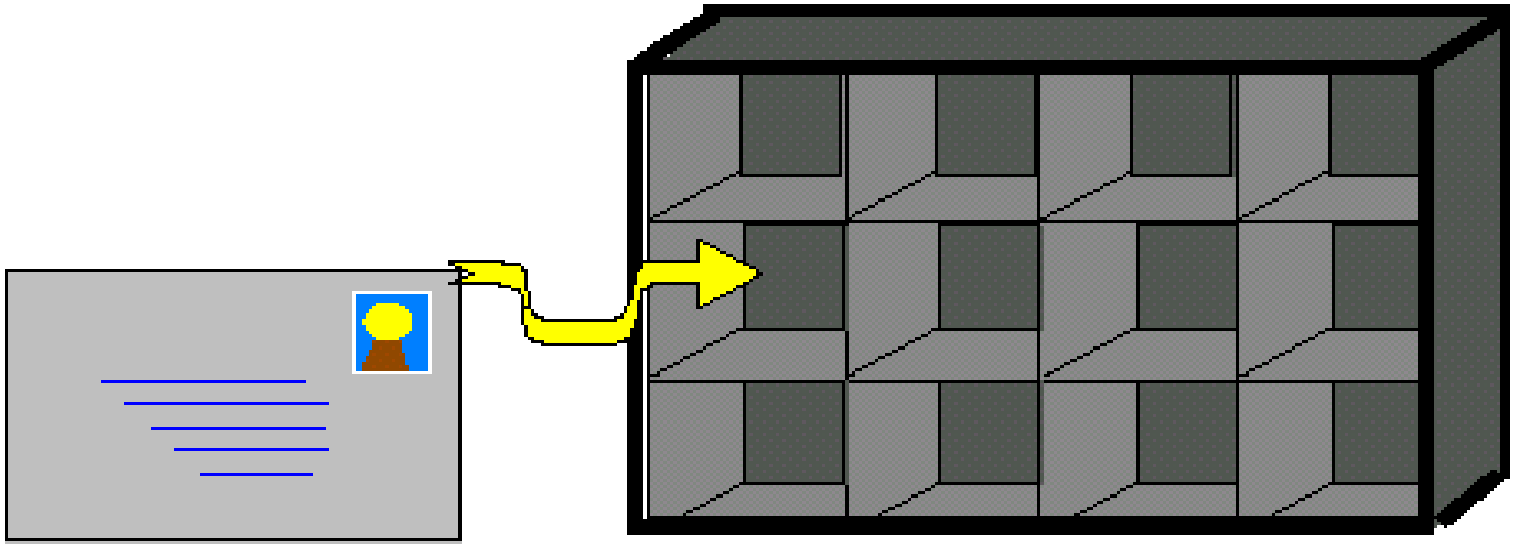
I/O and Peripherals: Examples

- Input Devices
  - Keyboard, Mouse, Digital Camera
- Output Devices
  - Monitor, Printer, Speaker
- Storage Peripherals
  - Magnetic Disks: hard disk
  - Optical Disks: CDROM, CD-RW, DVD
  - Flash Memory: pen drives

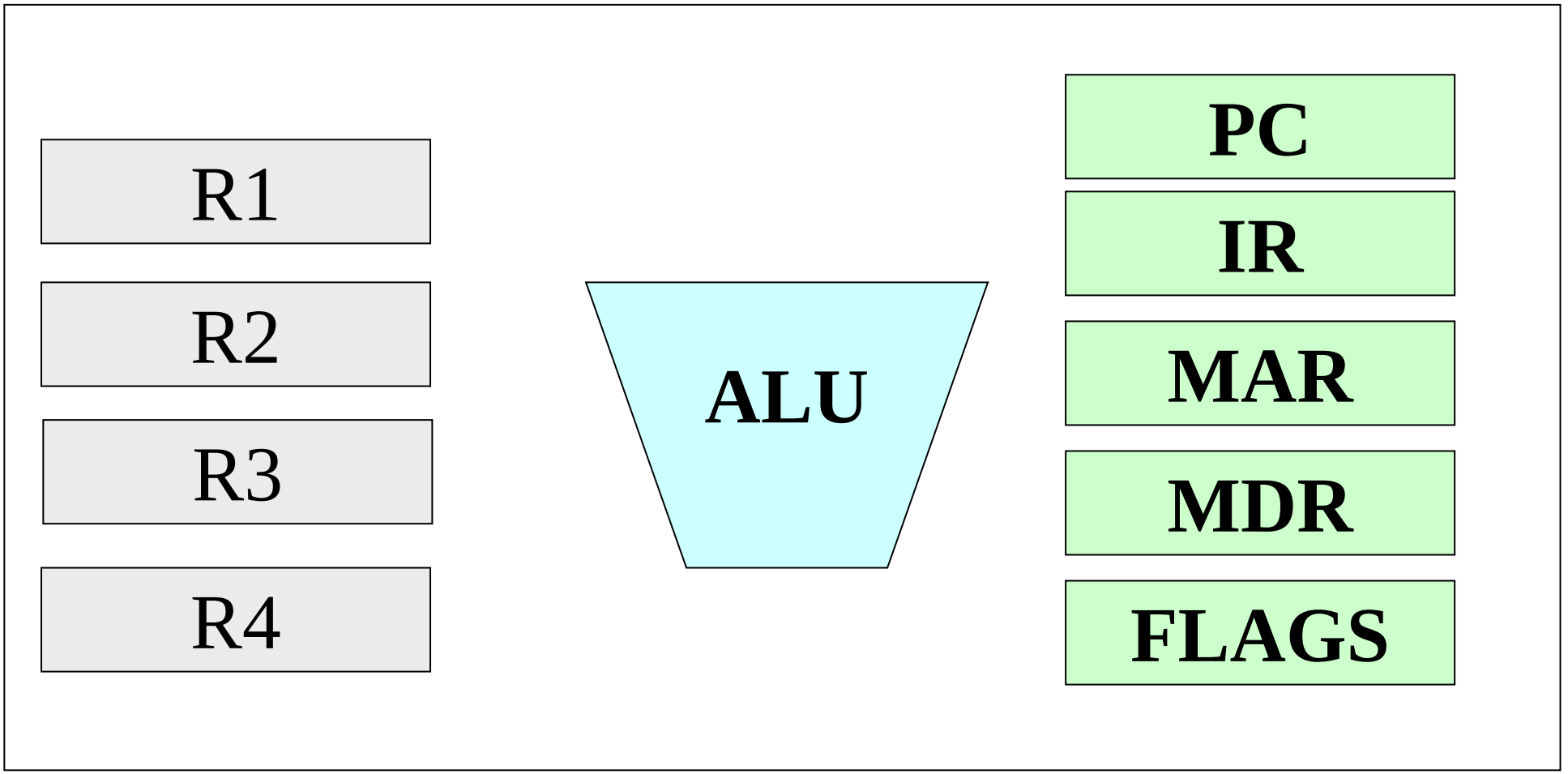
Memory: Address and Values

Every memory location has a **unique** address





CPU: A first cut



What can a computer do

- Determining if a given integer is a prime number
- A Palindrome recognizer
- Read in airline route information as a matrix and determine the shortest time journey between two airports
- Telephone pole placement problem
- Patriot Missile Control
- Finger-print recognition
- Chess Player
- Speech Recognition
- Language Recognition
- Discovering New Laws in Mathematics
- Automatic drug discovery
- …..

Programming and Software

Computer needs to be programmed to do such tasks

Programming is the 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 the instructions sequentially one after the other

Software: programs for doing tasks on computers

Contd.

- CPU understands machine language
  - Different strings of 0's and 1's only!!
  - Hard to remember and use
- Instruction set of a CPU
  - Mnemonic names for this strings

Instruction Set

- **Start**
- **Read M**
- **Write M**
- **Load Data, M**
- **Copy M1, M2**
- **Add M1, M2, M3**
- **Sub M1, M2, M3**
- **Compare M1, M2, M3**
- **Jump L**
- **J_Zero M, L**
- **Halt**

Instruction Set

- **Start**
- **Read M**
- **Write M**
- **Load Data, M**
- **Copy M1, M2**
- **Add M1, M2, M3**
- **Sub M1, M2, M3**
- **Compare M1, M2, M3**
- **Jump L**
- **J_Zero M, L**
- **Halt**

Program

0: Start

1: Load 33, 10

2: Load 24, 11

3: Add 10, 11, 12

4: Halt

Problems with programming using instruction sets directly

- Instruction sets of different types of CPUs different
  - Need to write different programs for computers with different types of CPUs even to do the same thing
- Still hard to remember
- Solution: High level languages (C, C++, Java,…)
  - CPU neutral, one program for many
  - Compiler to convert from high-level program to low level program that CPU understands

High-Level Program

```
Variables x, y;
Begin
Read (x);
Read (y);
If (x = y) then Write (x)
           else  Write (y);
End.
```

High-Level Program

```
Variables x, y;
Begin
Read (x);
Read (y);
If (x = y) then Write (x)
           else  Write (y);
End.
```

Low-Level Program

```
0: Start
1: Compare 20, 21, 22
2: J_Zero 22, 5
3: Write 20
4: Jump 6
5: Write 21
6: Halt
```

Three steps in writing programs

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

Step 2: Compile the program using a C compiler

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

Binary Representation

- 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
- 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: $723 = 3x10^0 + 2x10^1 + 7x10^2$

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
- Max. integer that can represented
  - in 1 byte = 255 (=11111111)
  - In 4 bytes = 4294967295 (= 32 1's)
- No. of integers that can be represented in 1 byte = 256 (the integers 0, 1, 2, 3,….255)

Fundamentals of C

First C program – print on screen

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

More print

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

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;
}
```

Reading values from keyboard

```
#include <stdio.h>
int main()
{
        int num ;
        scanf ("%d", &num) ;
        printf (“No. of students is %d\n”, num);
        return 0;
}
```

Centigrade to Fahrenheit

```
#include <stdio.h>
int main()
{
    float cent, fahr;
    scanf("%f",&cent);
    fahr = cent*(9.0/5.0) + 32;
    printf( "%f C equals %f F\n", cent, fahr);
    return 0;
}
```

Largest of two numbers

```
#include <stdio.h>
int main()
{
    int x, y;
    scanf("%d%d",&x,&y);
    if (x>y) printf("Largest is %d\n",x);
    else printf("Largest is %d\n",y);
    return 0;
}
```

largest-1.c

What does this do?

```
#include <stdio.h>
int main()
{
        int x, y;
        scanf(“%d%d”,&x,&y);
        if (x>y) printf(“Largest is %d\n”,x);
        printf(“Largest is %d\n”,y);
        return 0;
}
```

largest-2.c

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:

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

A C program should not contain anything else

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
- Each function has statements (instructions) for declaration, assignment, condition check, looping etc.
- Statements are executed one by one

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

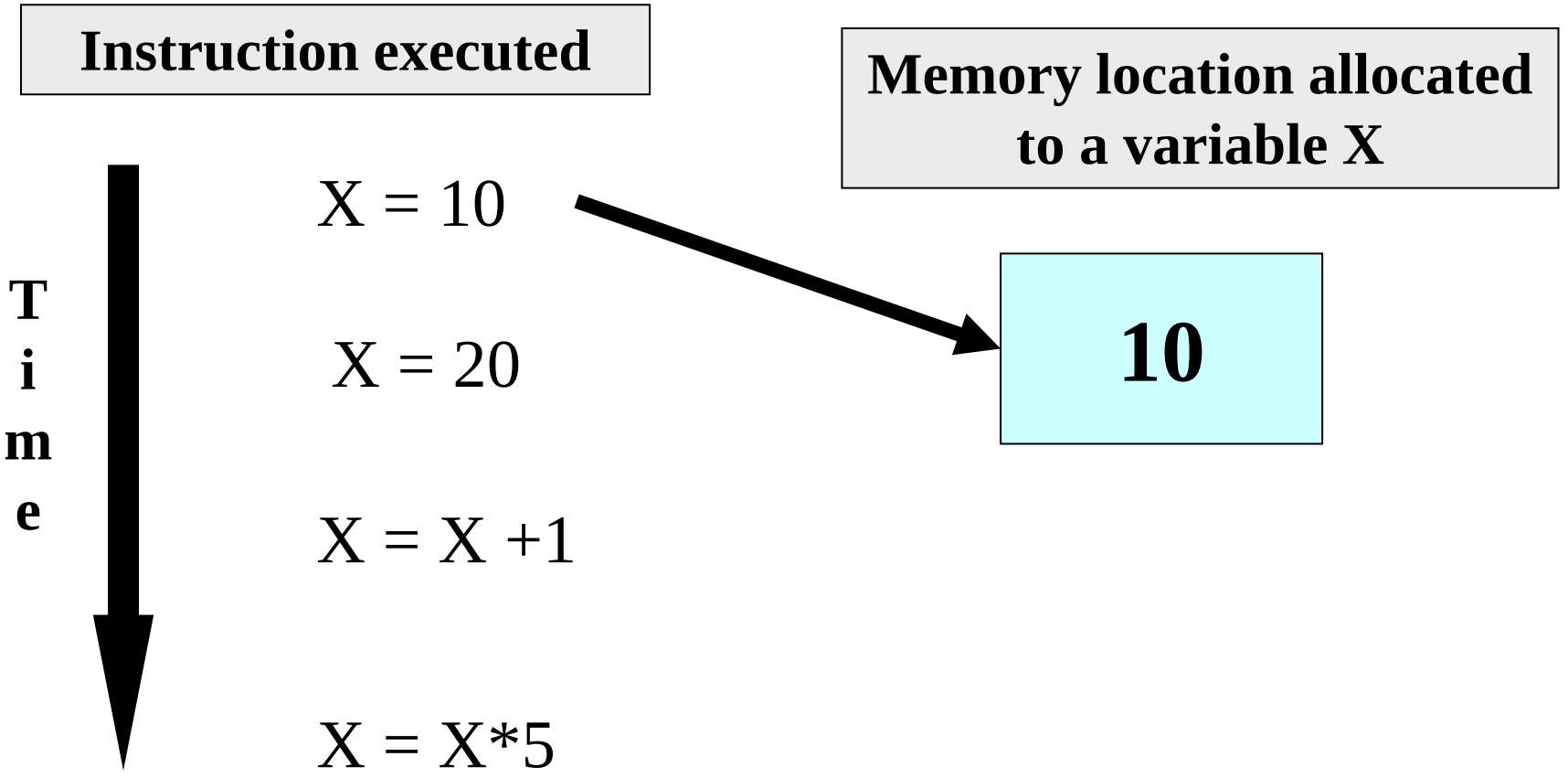
Contd.

- Variables stored in memory
- Remember that memory is a list of storage locations, each having a unique address
- A variable is like a bin
  - The contents of the bin is the value of the variable
  - The variable name is used to refer to the value of the variable
  - A variable is mapped to a location of the memory, called its address

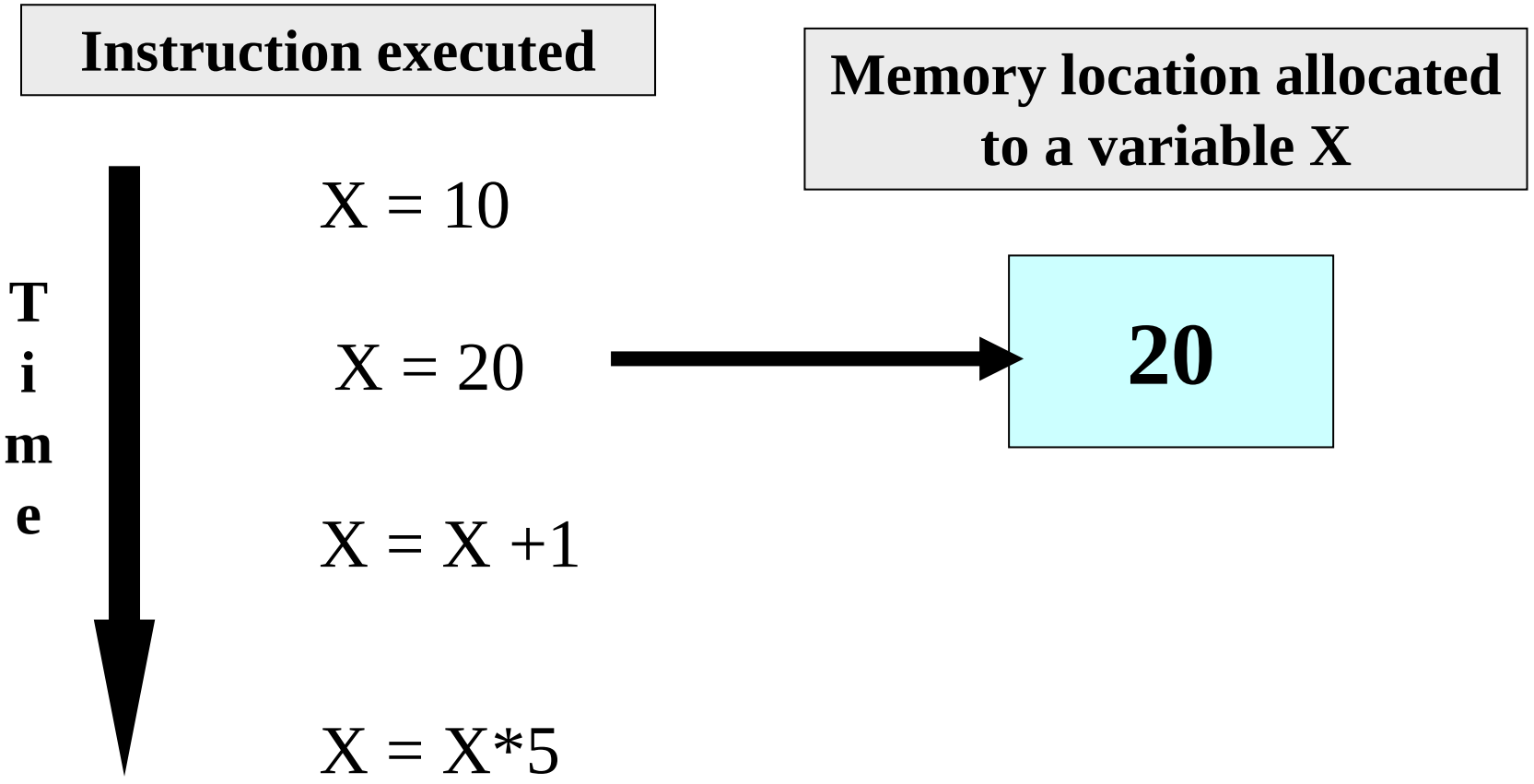
Example

```
#include <stdio.h>
int main( )
{
        int x;
        int y;
        x=1;
        y=3;
        printf("x = %d, y= %d\n", x, y);
         return 0;
}
```

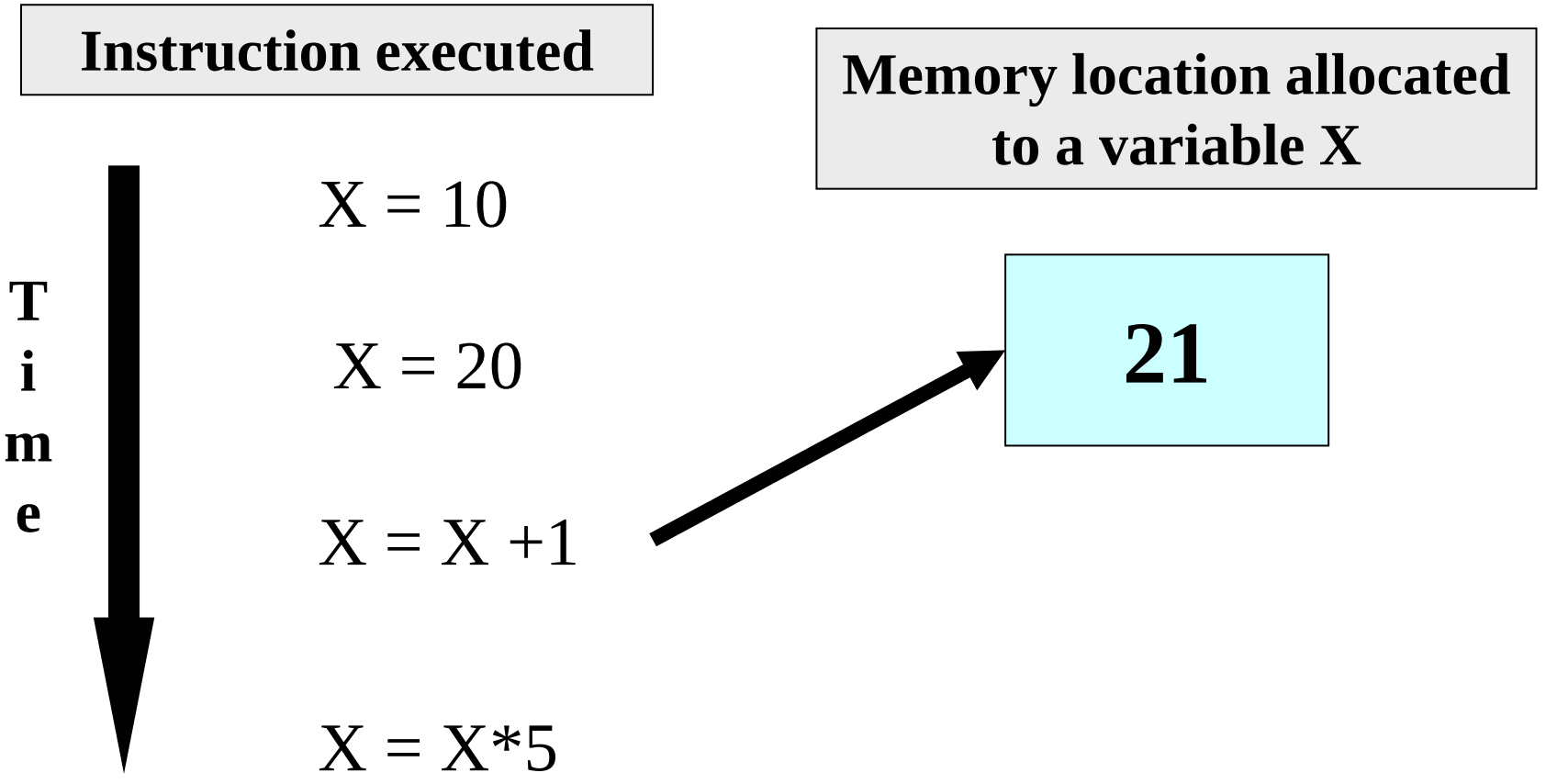
Variables in Memory



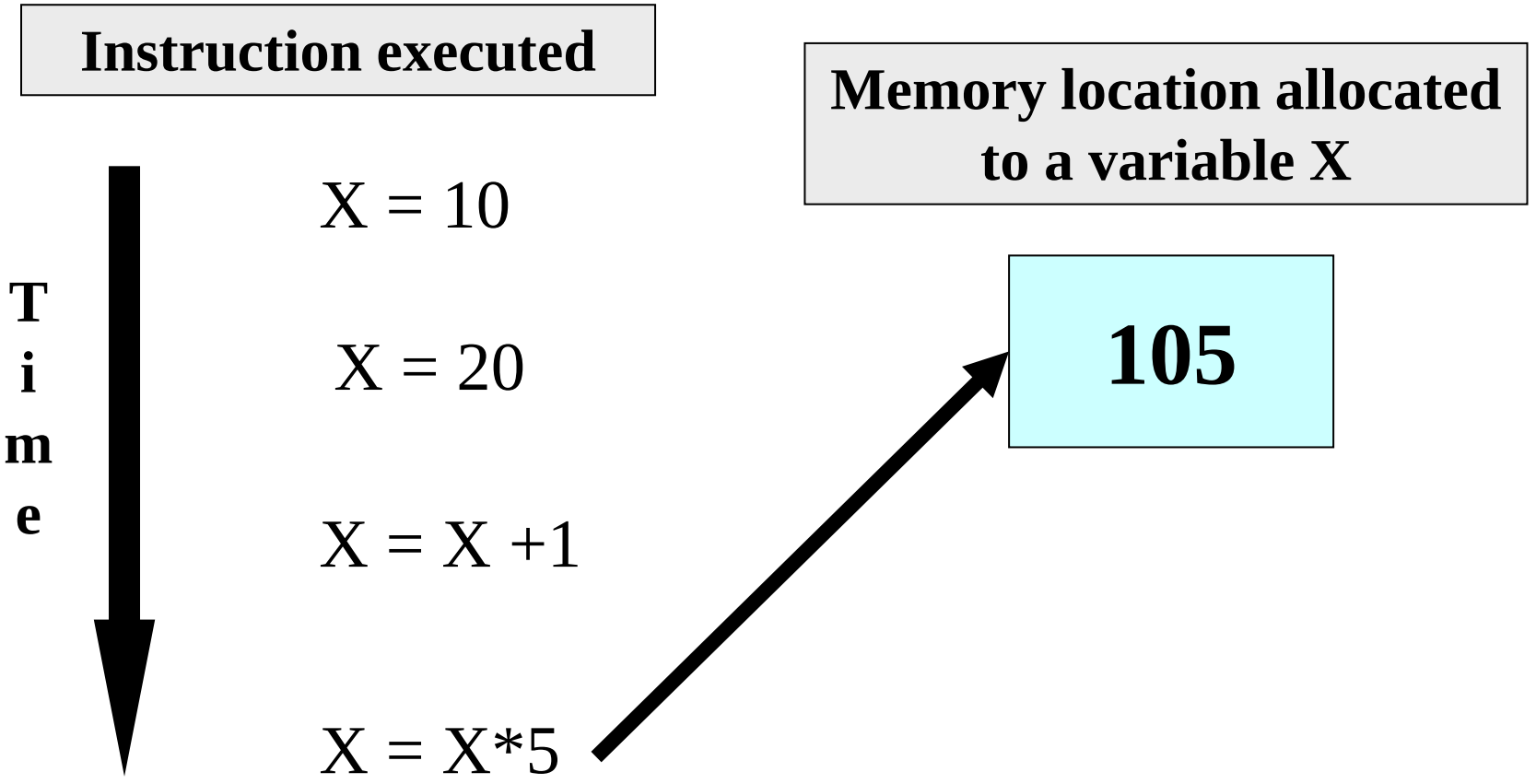
Variables in Memory



Variables in Memory



Variables in Memory



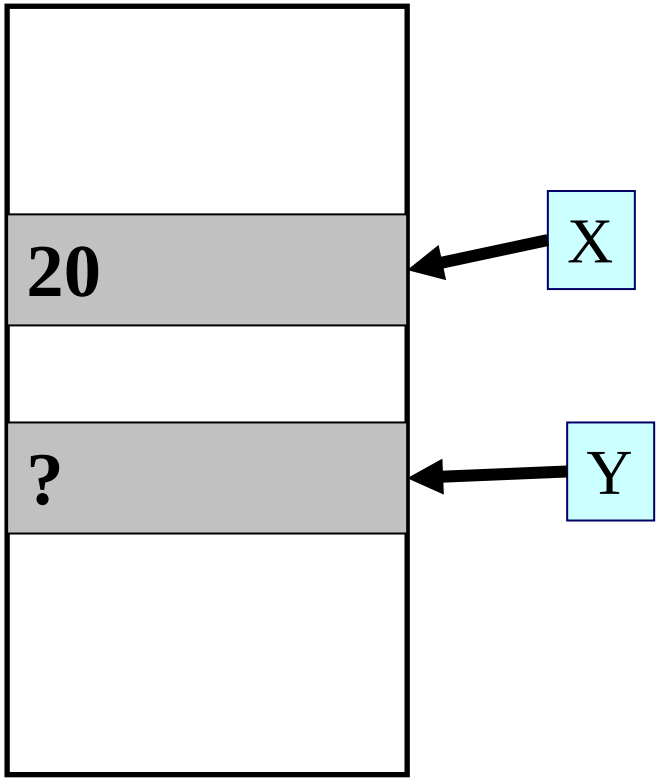
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



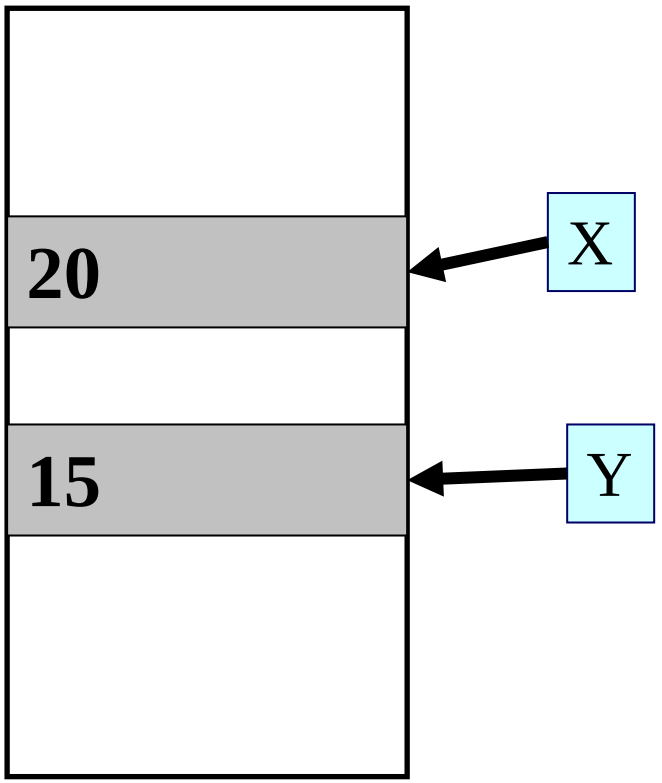
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



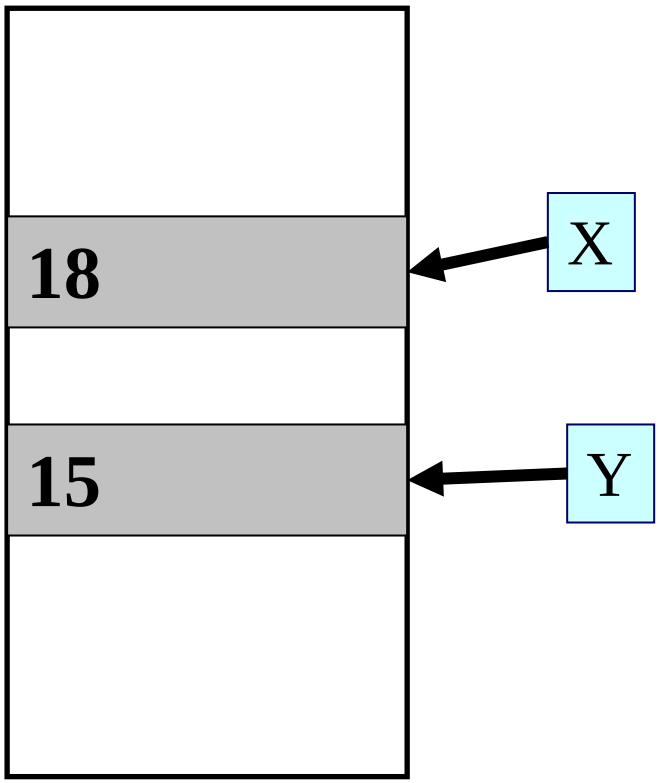
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



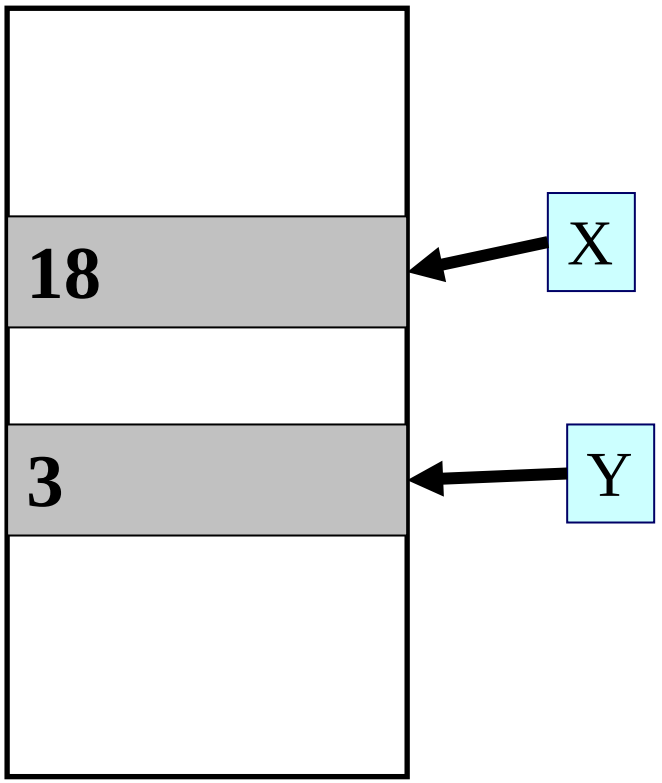
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



Data Types

- Each variable has a type, indicates what type of values the variable can hold
- Four common data types in C
  - 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)

Contd.

- Must declare a variable (specify its type and name) before using it anywhere in your program
- All variable declarations should be at the beginning of the main() or other functions
- A value can also be assigned to a variable at the time the variable is declared.

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

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

- Used by the C language, 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 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 int type variables declared

Values assigned

Example - 2

```
#include <stdio.h>
int main()
{
      float x, y;
      int d1, d2 = 10;
      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;
}
```

Assigns an initial value to d2, can be changed later

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
- 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”

Contd.

- Character constants
  - Contains a single character enclosed within a pair of single quote marks.
  - Examples :: ‘2’, ‘+’, ‘Z’
- Some special backslash characters

‘\n’	new line
‘\t’	horizontal tab
‘\’’	single quote
‘\”’	double quote
‘\\’	backslash
‘\0’	null

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 a 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) ;
scanf ("%c", &nextchar) ;
scanf ("%f", &length) ;
scanf (“%d%d”, &a, &b);
```

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

Contd.

- General syntax:

  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
  - Practice them in the lab