Data Structures and Programming Language

Arijit Mondal
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
arijit@iitp.ac.in

Introduction

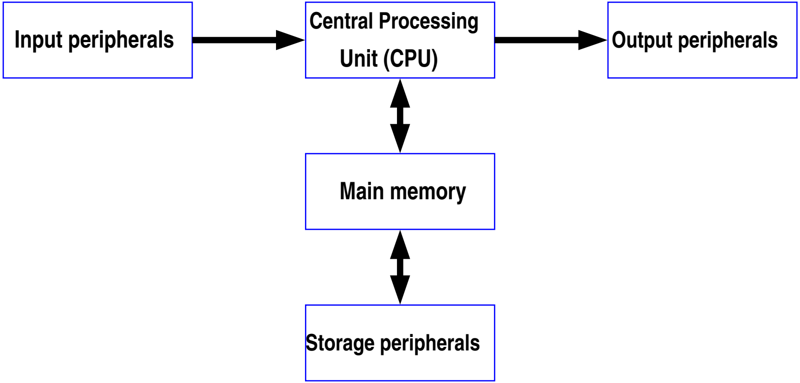
Course syllabus

- Introduction to C
- Variables, data type
- Statement, Conditional statement
- Loop construct
- Array, structure, union
- Function, Recursion
- Pointers
- Stack, queue, tree
- Searching, Sorting
- File handling

Books

- **Programming with C** by Byron Gottfried, Third Edition, Schaums Outlines Series,
- **The C Programming Language** by Brian W Kernighan, Dennis M Ritchie
- **Data structures** by S. Lipschutz, Schaums Outline Series

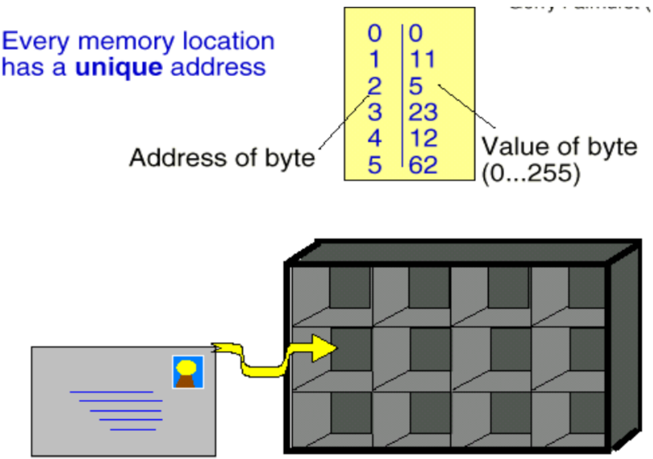
Simple view of computer



Peripherals

- Input devices
  - Keyboard, mouse, webcam
- Output devices
  - Monitor, printer, speaker
- Storage peripherals
  - Magnetic disks - hard disk
  - Optical disks - CDROM
  - Flash memory - pen drives

View of memory



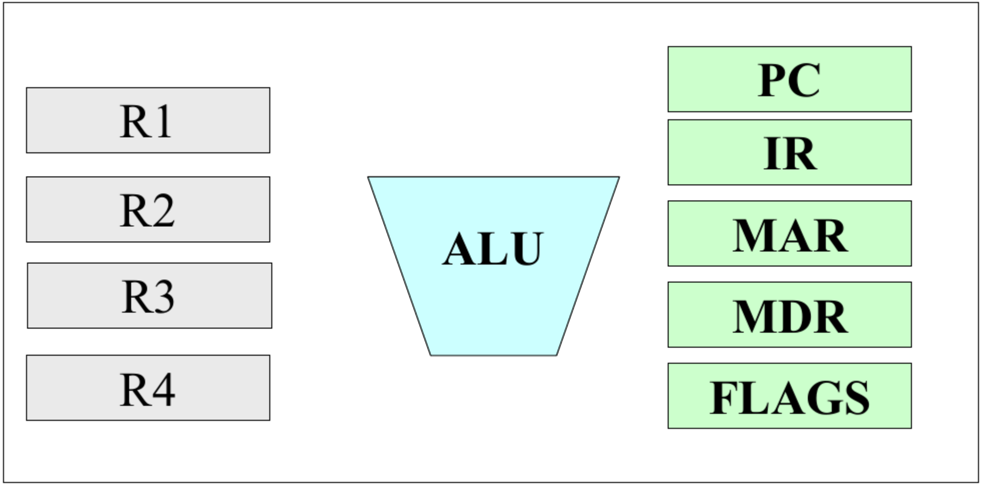
Every memory location has a **unique** address

0	0
1	11
2	5
3	23
4	12
5	62

Address of byte

Value of byte (0...255)

A simple view of CPU



What can a computer do

- Check prime number
- Palindrome recognizer
- Find shortest path between two points
- Telephone pole placement
- Spaceship control
- Finger-print recognition
- Play chess
- Speech recognition
- Language recognition and many more!

Program 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
- CPU understand **machine language**
  - Different strings of 0's and 1's only
  - Hard to remember and use
- **Instruction set** of CPU
  - Mnemonic names for this strings

Instruction set & Program

Instruction set

start

read m

write m

load data,m

copy m1,m2

add m1,m2,m3

compare m1,m2,m3

jump l

jz m,l

halt

Program

1. start
2. read 10
3. read 11
4. add 10,11,12
5. write 12
6. halt

Programming issue with instruction set

- 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 vs Low level program

```
variable x,y
begin
read x
read y
if(x>y) then write x
else write y
end
```

```
1. start
2. read 10
3. read 11
4. compare 10,11,12
5. jz 12,7
6. write 10
7. jump 9
8. write 11
9. 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)

Fundamentals of C

First C program

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

More print

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

More print

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

Reading values from keyboard

```
#include <stdio.h>
void main()
{
  int number;
  scanf("%d",&number);
  printf("Number of students in this class is %d\n",number);
}
```

Centigrade to Fahrenheit

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

Maximum of two numbers

```
#include <stdio.h>
void 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);
}
```

What will be the output?

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

The C character set

- C language alphabet
  - Uppercase letters 'A' to 'Z'
  - Lowercase letters 'a' to 'z'
  - Digits '0' to '9'
  - Special characters: ! # % ^& * - _ + = ~[ ] \| ; : ' " { } , . < > / ? blank
- **A C program should not contain anything else**

Structure of a C program

- A collection of functions
- Exactly one special function named **main** must be present.
  - Program always starts from there
- Each function has statements for declaration, assignment, condition check, looping, etc.
- Statements are executed one by one

Variables

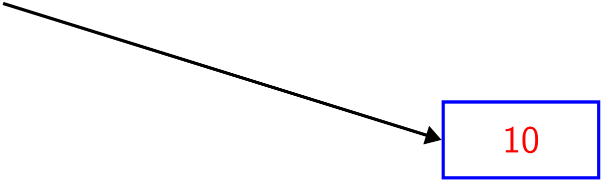
- Very important concept for programming
- An entity that has a value and is known to the program by a name
- Can store any temporary result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program
- 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>
void main()
{
  int x;
  int y;
  x=1;
  y=3;
  printf("x=%d, y=%d\n",x,y);
}
```

Variables in memory

$x = 10$



$x = 20$

$x = x + 1$

$x = x * 5$

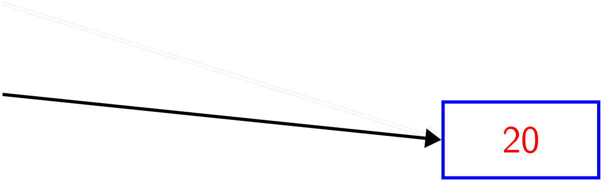
Variables in memory

$$x = 10$$

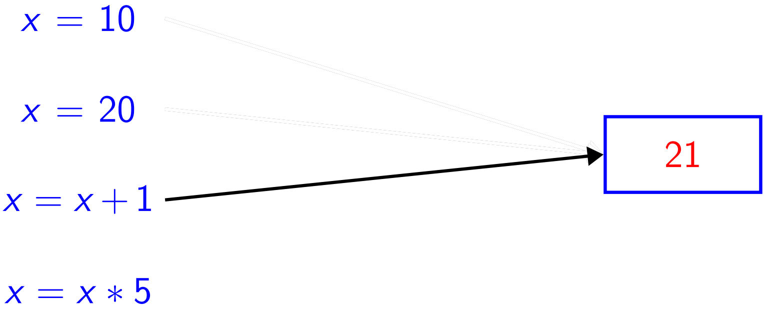
$$x = 20$$

$$x = x + 1$$

$$x = x * 5$$



Variables in memory



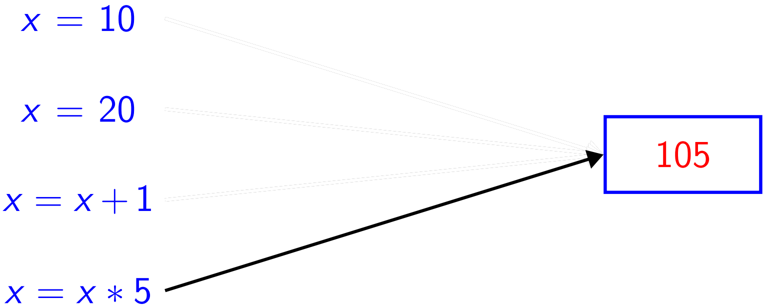
$x = 10$

$x = 20$

$x = x + 1$

$x = x * 5$

Variables in memory



$x = 10$

$x = 20$

$x = x + 1$

$x = x * 5$

105

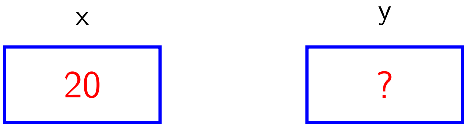
Variables in memory

$x = 20$

$y = 15$

$x = y + 3$

$y = x/6$



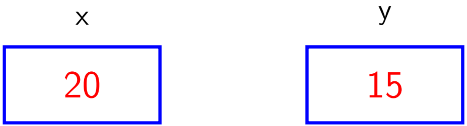
Variables in memory

$x = 20$

$\boldsymbol{y = 15}$

$x = y + 3$

$y = x/6$



x

20

y

15

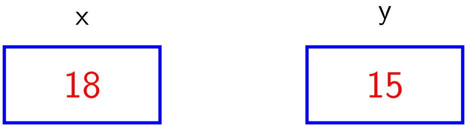
Variables in memory

$$x = 20$$

$$y = 15$$

$$x = y + 3$$

$$y = x/6$$



x

18

y

15

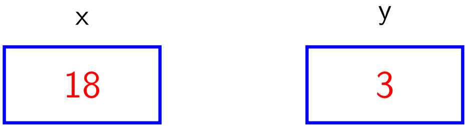
Variables in memory

$x = 20$

$y = 15$

$x = y + 3$

$y = x/6$



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)
- 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

Variable names

- 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>
void main()
{
  int x,y,sum;
  scanf("%d%d",&x,&y);
  sum=x+y;
  printf("Summation of x=%d and y=%d is %d\n",x,y,sum);
}
```

Example 2

```
#include <stdio.h>
void main()
{
  float x,y;
  int d1,d2;
  scanf("%f%f",&x,&y);
  printf("Summation of x=%f and y=%f is %f\n",x,y,x+y);
  printf(%d minus %d is %d\n",d1,d2,d1-d2);
}
```

Read only variable

- 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

Read only variable

- Correct

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

- Incorrect

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

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"
- 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);
```

Input: scanf function (contd.)

- 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", &temperature);`
  - `scanf("%lf", &length);`
  - `scanf("%d%d", &x, &y);`

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: `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 of printf

```
printf("Average of %d and %d is %f", a, b, avg);
printf("Hello!  \n Good Afternoon\n");
printf("%3d %5d %7d", a, b, a*a+b*b);
printf("%7.2f %5.1f", a, b);
```

Many more options are available for both printf and scanf. Read from book.