



CS10003: Programming & Data Structures

Spring 2021

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



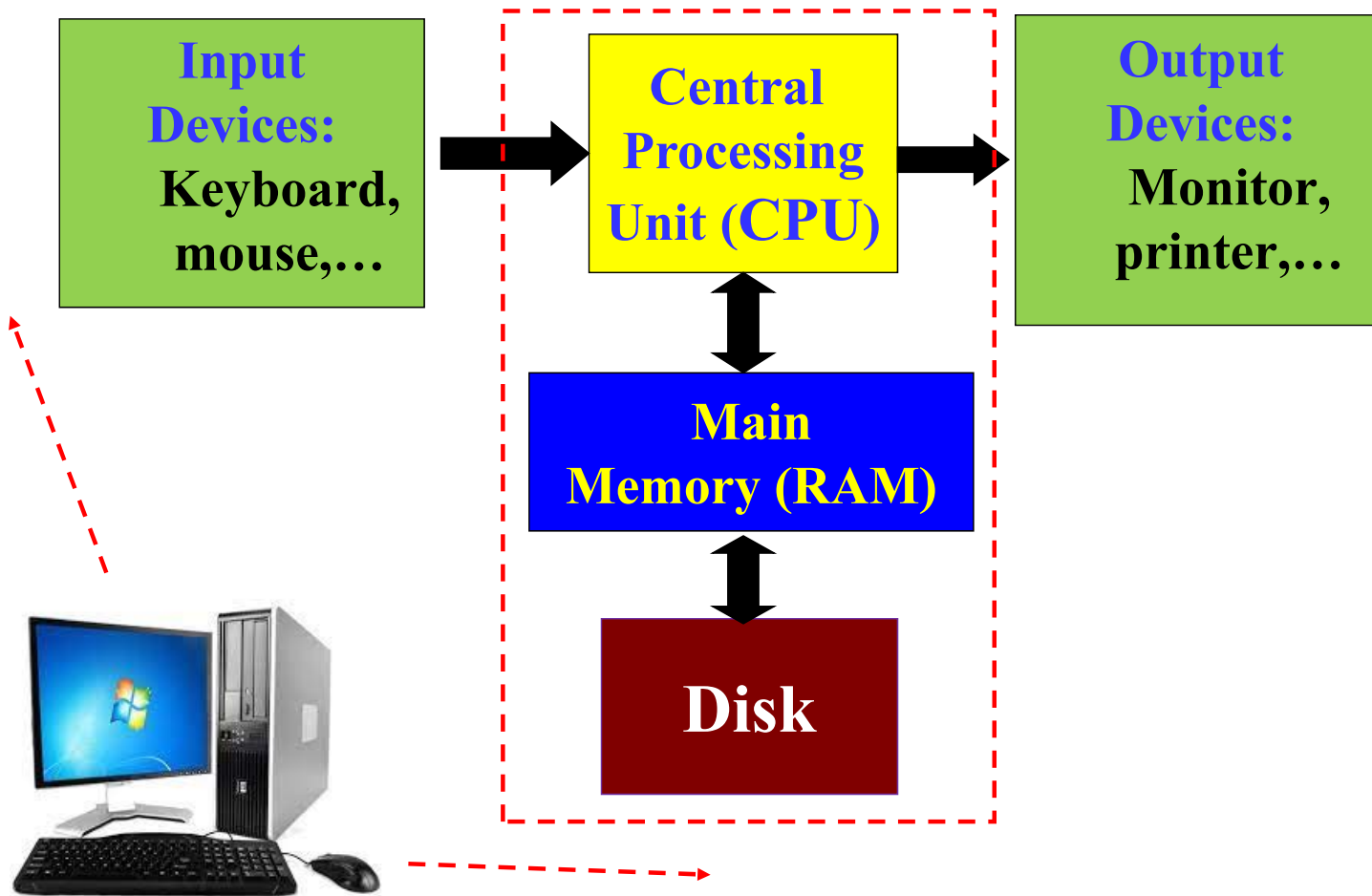
Teachers and Class Timings

- Section 11, 12
 - Monday (3-4:55 pm), Tuesday (3-3:55 pm)
 - Teacher: Prof. Sandip Chakraborty (SC)
- Section 13, 14
 - Monday (3-4:55 pm), Tuesday (3-3:55 pm)
 - Teacher: Prof. Shamik Sural (SS)
- Section 15, 16
 - Monday (3-4:55 pm), Tuesday (3-3:55 pm)
 - Teacher: Prof. Soumya Kanti Ghosh(SKG)
- Section 17, 18
 - Wednesday (10-10:55 am), Thursday (9-9:55 am), Friday (11-11:55 am)
 - Teacher: Prof. Bivas Mitra (BM)
- Section 19, 20
 - Wednesday (10-10:55 am), Thursday (9-9:55 am), Friday (11-11:55 am)
 - Teacher: Prof. Rajib Mall (RM)



Introduction

Basic Components of a Computer



Programming and Software



Computer needs to be **programmed** to do tasks...

Programming: Writing instructions in a **language** that can be understood by the computer --- so that it can perform a desired task.

Program: A sequence of instructions to do a task --- computer processes these instructions sequentially one after the other

Software: Commercial programs.

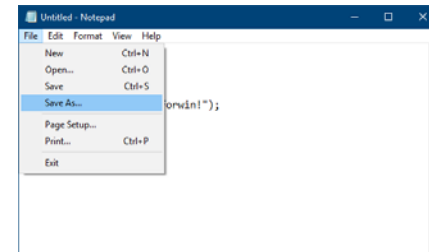
```
int a[] = {5, 4, 3, 2, 1}; // 20 bytes
const int n = sizeof(a) / sizeof(int);
for (int i = 0; i < n; i++)
    cout << a[i] << ' ';
cout << '\n';
sort(a, n);
```

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 (the computer executes it)

```
abhishek@itsfoss:~/Documents/Programs$ cat my_program.c
#include <stdio.h>
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
abhishek@itsfoss:~/Documents/Programs$ gcc -o my_program my_program.c
abhishek@itsfoss:~/Documents/Programs$ ./my_program
Hello, World!
```

Binary Representation

- Numbers are represented inside computers in the base-2 system (Binary Numbers)
 - Only two symbols/digits 0 and 1 **Example:1011**
 - Positional weights of digits: $2^0, 2^1, 2^2, \dots$ from right to left for integers
- Decimal number system we use is base-10
 - 10 digits, from 0 to 9, Positional weights $10^0, 10^1, 10^2, \dots$ from right to left for integers
 - Example: **$723 = 3 \times 10^0 + 2 \times 10^1 + 7 \times 10^2$**

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 be 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;
}
```

Output

Hello, World!

Parts of Our First C Program: *Hello World*

Header

```
#include <stdio.h>
```

Comments are good

```
/* This program prints "Hello World" */
```

main() means "start here"

```
main()
```

```
{
```

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

```
}
```

Brackets define code blocks

Library command

A Simple C program

```
#include <stdio.h>
int main()
{
    int x, y, sum, max;
    printf(" Enter x and y\n");
    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

Enter x and y

Output after you type 15 and 20

15 20
Sum = 35
Larger = 20

Structure of a C program

- A collection of **functions** (we will see those 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 statements for variable declarations, assignment, condition check, looping etc.
- Statements are executed one by one in order

```
int main()  
...  
}  
  
int f1()  
...  
}  
  
int f1()  
...  
}
```


Anatomy of a 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;
```

```
}
```

main function

Declaration statement

Input statement

Assignment statements

Control statement

Output statement

Return statement

Writing a C program

- You have to first understand what different statements do --- to decide which ones you should use and 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 ...
 - Do not question at the moment why you have to type exactly like this, you just have to or it is not a C program!!
 - Compiler will give error if your typed program does not match required C syntax
- There are other rules to follow

```
#63 <!DOCTYPE html>  
SyntaxError: invalid  
syntax
```



Things you might need 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 any other characters

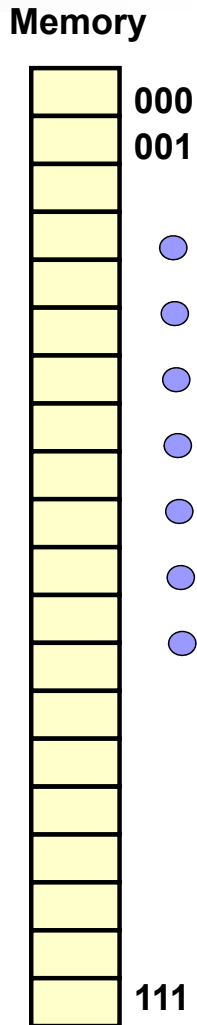


Variables

- Very important concept in programming
- It is an entity that has a value and is known to the program by its name **a=b+c;**
- 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
- Variables are stored in memory
- The value of a variable can be changed during the execution of the program

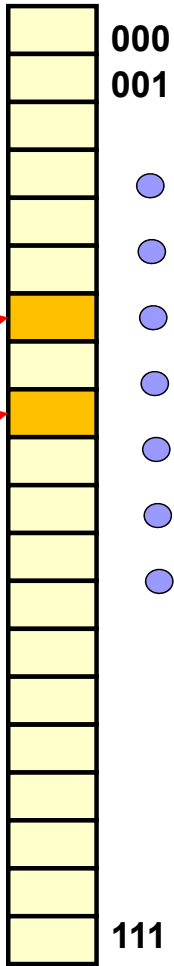
Contd.

- Variables are stored in memory
- Memory is a list of consecutive storage locations, each having a unique address
- A variable is like a **bin**
 - The content 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

Instruction executed

Memory location allocated
to a variable X

T
i
m
e

$X = 10$

$X = 20$

$X = X + 1$

$X = X * 5$

10

Variables in Memory

Instruction executed

Memory location allocated
to a variable X

T
i
m
e

$$X = 10$$

$$X = 20$$

$$X = X + 1$$

$$X = X * 5$$

20

Variables in Memory

Instruction executed

Memory location allocated
to a variable X

T
i
m
e

$$X = 10$$

$$X = 20$$

$$X = X + 1$$

$$X = X * 5$$

21

Variables in Memory

Instruction executed

Memory location allocated
to a variable X

T
i
m
e

$$X = 10$$

$$X = 20$$

$$X = X + 1$$

$$X = X * 5$$

105

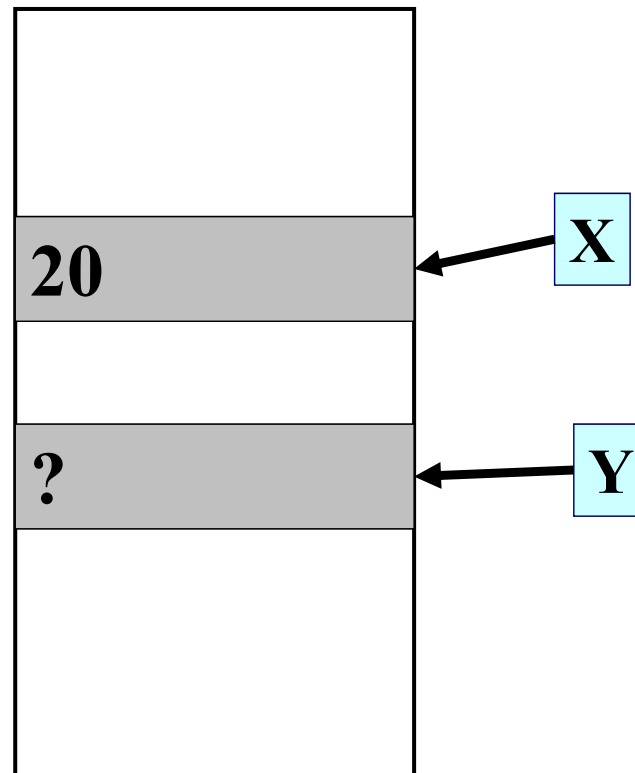
Variables (contd.)

X = 20

Y=15

X = Y+3

Y=X/6



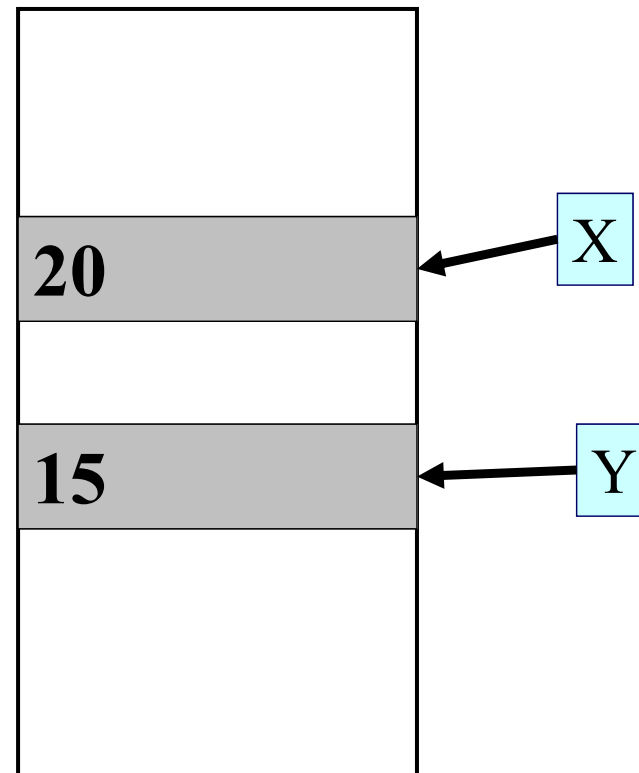
Variables (contd.)

$$X = 20$$

$$Y = 15$$

$$X = Y + 3$$

$$Y = X / 6$$



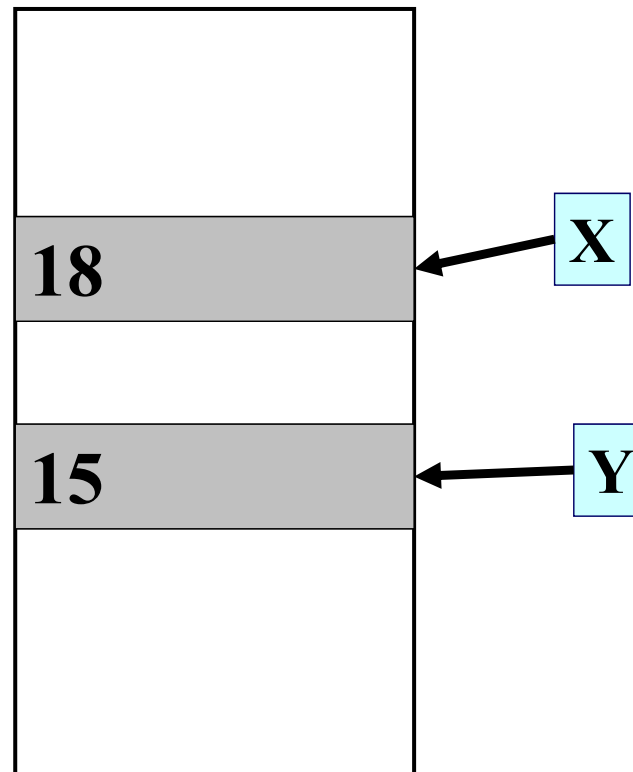
Variables (contd.)

$$X = 20$$

$$Y = 15$$

$$X = Y + 3$$

$$Y = X / 6$$



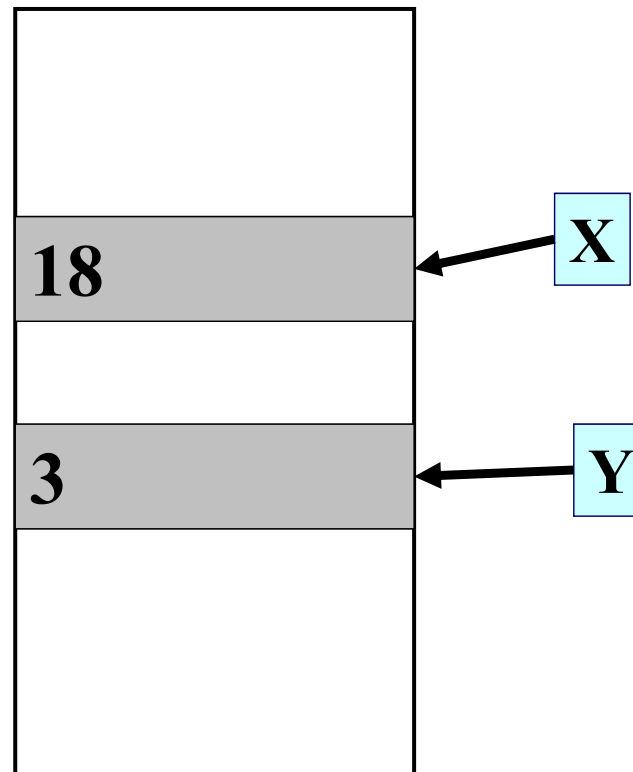
Variables (contd.)

$$X = 20$$

$$Y = 15$$

$$X = Y + 3$$

$$Y = X / 6$$





Data Types

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

Contd.

- 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;
```

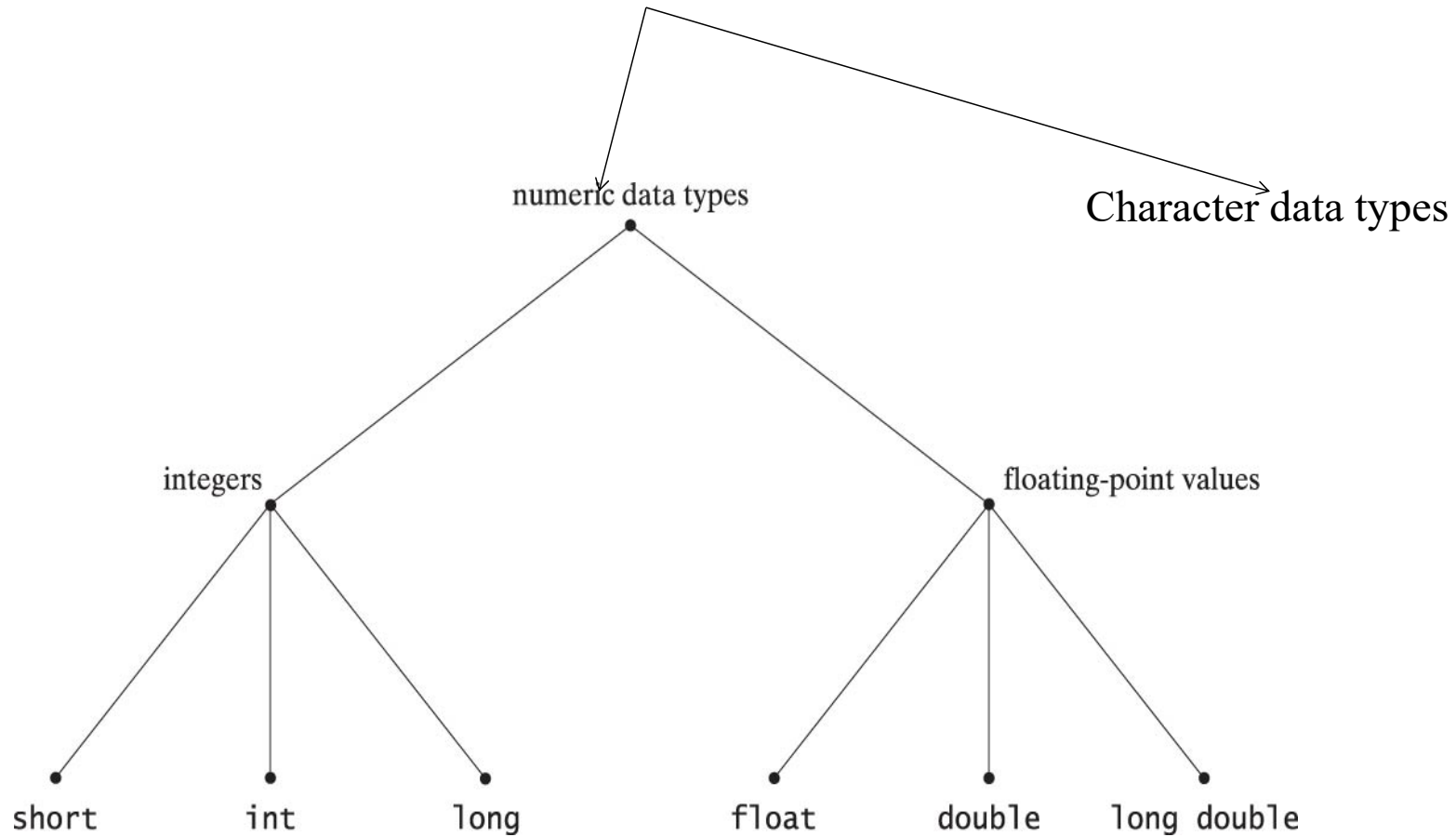
```
char flag = 'y';
```

```
#include <stdio.h>
int main( )
{
    int x;
    int y;
    x=1;
    y=3;
    return 0;
}
```

Data types

- Three common data are typically types used:
 - **Integer** :: can store only whole numbers
 - Examples: 25, -56, 1, 0
 - **Floating-point** :: can store numbers with fractional values.
 - Examples: 3.14159, 5.0, -12345.345
 - **Character** :: can store a character
 - Examples: 'A', 'a', '*', '3', ' ', '+'

C Data Types



Variable Names

- Sequence of letters and digits
- First character must either 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 2sqrt class,rank a&b**

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: Add Two Numbers

```
#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;
```

Assigns an initial value to d2,
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 (Constants)

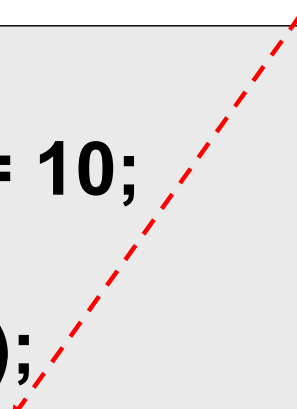
- Variables whose values can be initialized during declaration, but cannot be changed after that
- Declared by putting the **const** keyword in front of the declaration
- Storage allocated just like any variable
- Used for variables whose values need not be changed
 - Prevents accidental change of the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

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

Correct

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:

e means “10 to the power of”

- Decimal notation: **25.0, 0.0034, .84, -2.234**
- Exponential (scientific) notation

3.45e23, 0.123e-12, 123e2

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

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
 - 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 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;      Variable list (note the &  
float c;    before a variable name)  
scanf(“%d%d%f”, &a, &b, &c);  
      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);
```

Examples Explained

■ `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), or 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 (usually 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 format options are available for both printf and scanf

- Read from the book

- Practice them in your computer!**



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 = %5.2f\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?

Some more print

```
#include <stdio.h>
int main()
{
    char c1, c2;
    scanf("%c%c", &c1, &c2);
    printf("%c%c", c1, c2);
    return 0;
}
```

Output

```
ab
ab
```

What about this?

```
#include <stdio.h>
int main()
{
    char c1, c2;
    scanf("%c%c", &c1, &c2);
    printf("%c%c", c1, c2);
    return 0;
}
```

Output

```
a b
a
```

Can you explain why only 'a' was printed this time, even though it is the same program as in the last slide? Note the difference from the last slide carefully. Also note that two characters were read this time also, or scanf would have waited

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.