CS11001/CS11002
Programming and Data Structures (PDS)

(Theory: 3-1-0)

The basic components of a digital computer

- **Input devices** These are the devices using which the user provides input instances. In a programmable computer, input devices are also used to input programs. Examples: keyboard, mouse.
- **Output devices** These devices notify the user about the outputs of a computation. Example: screen, printer.

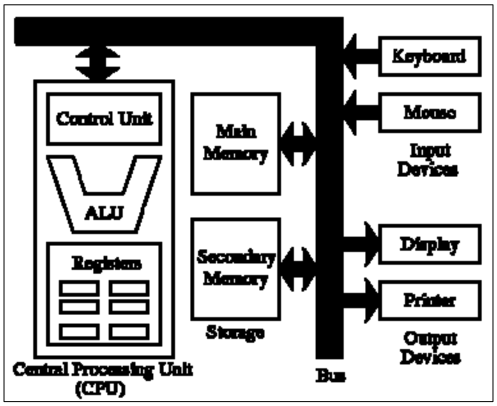
Processing unit

- The central processing unit (**CPU**) is the *brain* of the computing device and performs the basic processing steps. A CPU typically consists of:
  - **An arithmetic and logical unit (ALU):** This provides the basic operational units of the CPU. It is made up of units (like adders, multipliers) that perform arithmetic operations on integers and real numbers, and of units that perform logical operations (logical and bitwise AND, OR etc.).
  - **A control unit:** This unit is responsible for controlling flow of data and instructions.
  - **General purpose registers:** A CPU usually consists of a finite number of memory cells that work as scratch locations for storing intermediate results and values.

External memory

- The amount of memory (registers) resident in the CPU is typically very small and is inadequate to accommodate programs and data even of small sizes. Out-of-the-processor memory provides the desired storage space. External memory is classified into two categories:
  - **Main (or primary) memory:** This is a high-speed memory that stays *close* to the CPU. Programs are first loaded in the main memory and then executed. Usually main memory is volatile, i.e., its contents are lost after power-down.
  - **Secondary memory:** This is relatively inexpensive, bigger and low-speed memory. It is normally meant for off-line storage, i.e., storage of programs and data for future processing. One requires secondary storage to be permanent, i.e., its contents should last even after shut-down. Examples of secondary storage include floppy disks, hard disks and CDROM disks.

The von Neumann architecture



- John von Neumann proposed the first usable draft of a working computer.

How does a program run in a computer?

- The inputs, the intermediate values and the instructions defining the processing stage reside in the (main) memory.
- **Data area:** The data area stores the variables needed for the processing stage.
  - The values stored in the data area can be read, written and modified by the CPU.
  - The data area is often divided into two parts:
    - a stack part : It typically holds all statically allocated memory (global and local variables),
    - a heap part: It is used to allocate dynamic memory to programs during run-time.

Instruction area

- The instruction area stores a sequence of instructions that define the steps of the program.
- Under the control of a clock, the computer carries out a **fetch-decode-execute** cycle:
  - in which instructions are fetched one-by-one from the instruction area to the CPU
  - decoded in the control unit
  - and executed in the ALU.

  Instruction Set Architecture (ISA): *The CPU understands only a specific set of instructions. The instructions stored in memory must conform to this specification.*

The fetch-decode-execute cycle

- A sequence of machine instructions is copied to the instruction area of the memory.
- Some global variables and input parameters are copied to the data area of the memory.
- A particular control register, called the **program counter** (**PC**), is loaded with the address of the first instruction of the program.
- The CPU fetches the instruction from that location in the memory that is currently stored in the PC register.

The fetch-decode-execute cycle

- The instruction is decoded in the control unit of the CPU.
- The instruction may require one or more operands.
  - An operand may be either a data or a memory address.
    - A data may be either a constant (also called an immediate operand) or a value stored in the data area of the memory or a value stored in a register.
    - An address may be either immediate or a resident of the main memory or available in a register.

The fetch-decode-execute cycle

- An immediate operand is available from the instruction itself. The content of a register is also available at the time of the execution of the instruction.
- Finally, a variable value is fetched from the data part of the main memory.

The fetch-decode-execute cycle

- If the instruction is a data movement operation, the corresponding movement is performed.
  - a "load" instruction copies the data fetched from memory to a register
  - a "store" instruction sends a value from a register to the data area of the memory.
- If the instruction is an arithmetic or logical instruction, it is executed in the ALU after all the operands are available in the CPU (in its registers). The output from the ALU is stored back in a register.

The fetch-decode-execute cycle

- If the instruction is a jump instruction, the instruction must contain a memory address to jump to.
- The program counter (PC) is loaded with this address.
  - A jump may be conditional, i.e., the PC is loaded with the new address if and only if some condition(s) is/are true.
- If the instruction is not a jump instruction, the address stored in the PC is incremented by one.
- If the end of the program is not reached, the CPU continues its fetch-decode-execute cycle.

Back to C Programs

Example 3

```
#include <stdio.h>
main ()
{
   int n;
   scanf("%d",&n);
   printf("%d\n",n*n);
}
```

Example 4

```
#include <stdio.h>
main ()
{
   int n;
   scanf("%d",&n);
    printf("%d\n",1/n);
}
```

Example 5

```
#include <stdio.h>
main ()
 {
   int n;
   scanf("%d",&n);
   printf("%f\n",1.0/n);
}
```

Character Sets in C

- Alphabets: A, B, …, Z

  a, b, …, z
- Digits: 0, 1, …9
- Special Characters:

  , < > .; % \ | ~ # ? ( ) “ “ + etc.
- White Space Characters:

  blank space, newline, tab etc

Identifiers and Keywords

- Identifiers are used to identify or name variables.
- Identifiers names must be sequences of letters and digits, and must begin with a letter
- The underscore character ‘_’ is considered a letter
- Names should not be the same as a keyword like ‘int’, ‘char’, ‘void’ etc.
- C is case sensitive.
- For any internal identifier, at least the first 31 characters are significant in any ANSI C Compiler.

Variables

- A variable is an entity that has a value and is known to the program by a name.
- A variable definition associates a memory location with the variable name.
- A variable can have only one value assigned to it at any given time during the execution of the program.
- Its value gets updated/changed during the execution of the program.

Example: f= 1.8 * c + 32

Variable Names

- Sequence of letters and digits.
- First character is a letter.
- Examples: i, rank1, MAX, Min, class_rank
  dataType
- Invalid examples:
  a’s, fact rec, 2sqroot
  class,rank

Data Types

- C language supports the following basic data types:

  char: a single byte that can hold one character

  int: an integer

  float: a single precision floating point number

  double: a double precision floating point number

Precision refers to the number of significant digits after the decimal point.

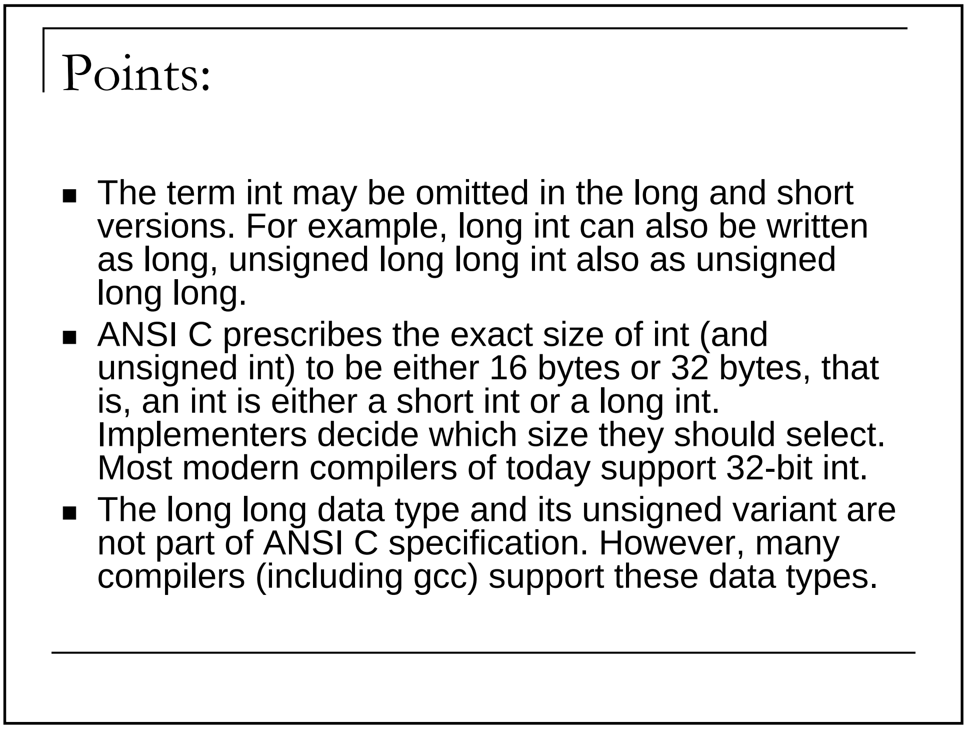
Data Types

- Abstraction is necessary.

- Integer Data Types:

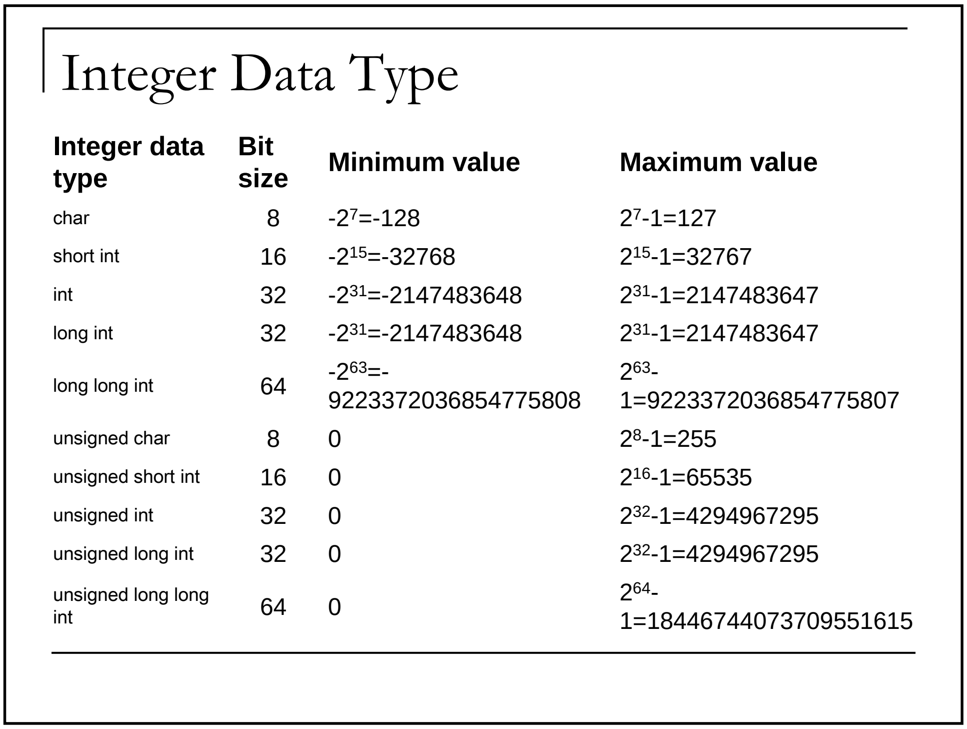
  Integers are whole numbers that can assume both positive and negative values, i.e., elements of the set:

  { ..., -3, -2, -1, -, 1, 2, 3, ... }



Points:

- The term int may be omitted in the long and short versions. For example, long int can also be written as long, unsigned long long int also as unsigned long long.
- ANSI C prescribes the exact size of int (and unsigned int) to be either 16 bytes or 32 bytes, that is, an int is either a short int or a long int. Implementers decide which size they should select. Most modern compilers of today support 32-bit int.
- The long long data type and its unsigned variant are not part of ANSI C specification. However, many compilers (including gcc) support these data types.



Integer Data Type

Integer data type	**Bit size**	**Minimum value**	**Maximum value**
char	8	$-2^{7}=-128$	$2^{7}-1=127$
short int	16	$-2^{15}=-32768$	$2^{15}-1=32767$
int	32	$-2^{31}=-2147483648$	$2^{31}-1=2147483647$
long int	32	$-2^{31}=-2147483648$	$2^{31}-1=2147483647$
long long int	64	$-2^{63}=-9223372036854775808$	$2^{63}-1=9223372036854775807$
unsigned char	8	0	$2^{8}-1=255$
unsigned short int	16	0	$2^{16}-1=65535$
unsigned int	32	0	$2^{32}-1=4294967295$
unsigned long int	32	0	$2^{32}-1=4294967295$
unsigned long long int	64	0	$2^{64}-1=18446744073709551615$

Float Data Type

- Like integers, C provides representations of real numbers and those representations are finite.
- Depending on the size of the representation, C's real numbers have got different names.

Real data type	Bit size
float	32
double	64
long double	128

Char data type

- char for representing characters.
- We need a way to express our thoughts in writing.
- This has been traditionally achieved by using an alphabet of symbols with each symbol representing a sound or a word or some punctuation or special mark.
- The computer also needs to communicate its findings to the user in the form of something written.

Char data type

- Since the outputs are meant for human readers, it is advisable that the computer somehow translates its bit-wise world to a human-readable script.
- The Roman script (mistakenly also called the English script) is a natural candidate for the representation.
  - The Roman alphabet consists of the lower-case letters (a to z), the upper case letters (A to Z), the numerals (0 through 9) and some punctuation symbols (period, comma, quotes etc.).
  - In addition, computer developers planned for inclusion of some more control symbols (hash, caret, underscore etc.). Each such symbol is called a **character**.

ASCII Code

- In order to promote interoperability between different computers, some standard encoding scheme is adopted for the computer character set.
- This encoding is known as **ASCII** (abbreviation for **American Standard Code for Information Interchange**).
- In this scheme each character is assigned a unique integer value between 32 and 127.
- Since eight-bit units (bytes) are very common in a computer's internal data representation, the code of a character is represented by an 8-bit unit. Since an 8-bit unit can hold a total of 2^8=256 values and the computer character set is much smaller than that, some values of this 8-bit unit do not correspond to visible characters.

Printable Characters

- These values are often used for representing invisible control characters (like line feed, alarm, tab etc.) and extended Roman letters (inflected letters like ä, é, ç). Some values are reserved for possible future use. The ASCII encoding of the printable characters is summarized in the following table.

Decimal	Hex	Binary	Character	Decimal	Hex	Binary	Character
32	20	00100000	SPACE	80	50	01010000	P
33	21	00100001	!	81	51	01010001	Q
34	22	00100010	"	82	52	01010010	R
35	23	00100011	#	83	53	01010011	S
36	24	00100100	$	84	54	01010100	T
37	25	00100101	%	85	55	01010101	U
38	26	00100110	&	86	56	01010110	V
39	27	00100111	'	87	57	01010111	W
40	28	00101000	(	88	58	01011000	X
41	29	00101001	)	89	59	01011001	Y
42	2a	00101010	*	90	5a	01011010	Z
43	2b	00101011	+	91	5b	01011011	[
44	2c	00101100	,	92	5c	01011100	\
45	2d	00101101	-	93	5d	01011101	]
46	2e	00101110	.	94	5e	01011110	^
47	2f	00101111	/	95	5f	01011111	_
48	30	00110000	0	96	60	01100000	`
49	31	00110001	1	97	61	01100001	a
50	32	00110010	2	98	62	01100010	b

51	33	00110011	3	99	63	01100011	c
52	34	00110100	4	100	64	01100100	d
53	35	00110101	5	101	65	01100101	e
54	36	00110110	6	102	66	01100110	f
55	37	00110111	7	103	67	01100111	g
56	38	00111000	8	104	68	01101000	h
57	39	00111001	9	105	69	01101001	i
58	3a	00111010	:	106	6a	01101010	j
59	3b	00111011	;	107	6b	01101011	k
60	3c	00111100	<	108	6c	01101100	l
61	3d	00111101	=	109	6d	01101101	m
62	3e	00111110	>	110	6e	01101110	n
63	3f	00111111	?	111	6f	01101111	o
64	40	01000000	@	112	70	01110000	p
65	41	01000001	A	113	71	01110001	q
66	42	01000010	B	114	72	01110010	r
67	43	01000011	C	115	73	01110011	s
68	44	01000100	D	116	74	01110100	t
69	45	01000101	E	117	75	01110101	u
70	46	01000110	F	118	76	01110110	v

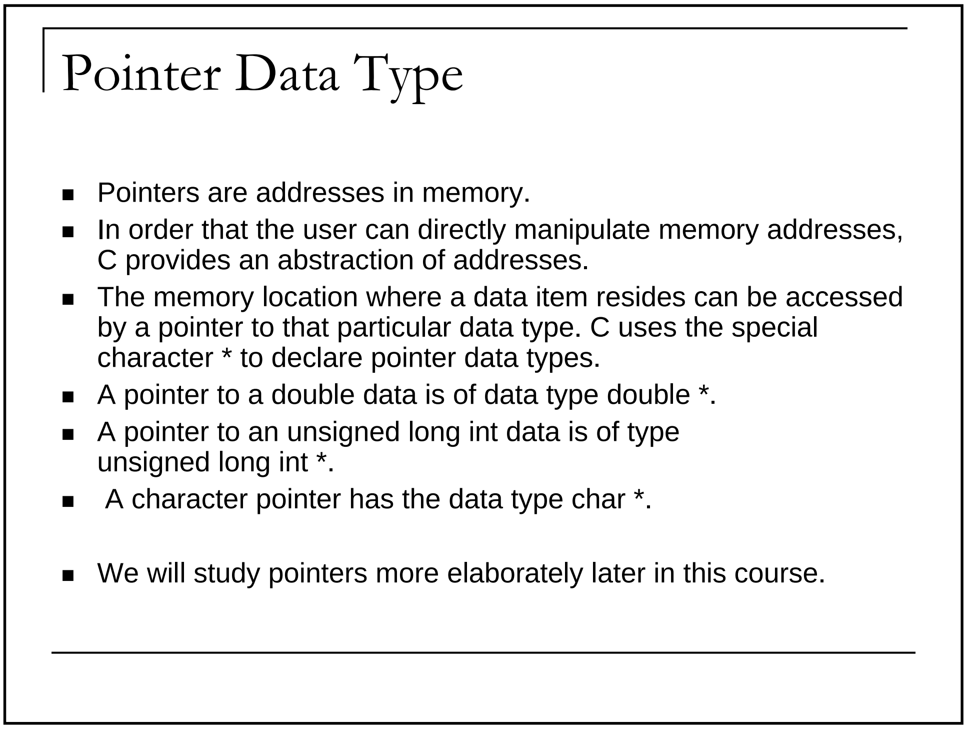
71	47	01000111	G		119	77	01110111	w
72	48	01001000	H		120	78	01111000	x
73	49	01001001	I		121	79	01111001	y
74	4a	01001010	J		122	7a	01111010	z
75	4b	01001011	K		123	7b	01111011	{
76	4c	01001100	L		124	7c	01111100	|
77	4d	01001101	M		125	7d	01111101	}
78	4e	01001110	N		126	7e	01111110	~
79	4f	01001111	O		127	7f	01111111	DELETE

Qualifiers

- Qualifiers add more data types.
  - typically *size* or *sign.*
  - *size: short or long*
  - *sign: signed or unsigned*
- *signed short int, unsigned short int, signed int, signed long int, long double, long int*
- *signed char and unsigned char*

A Comment

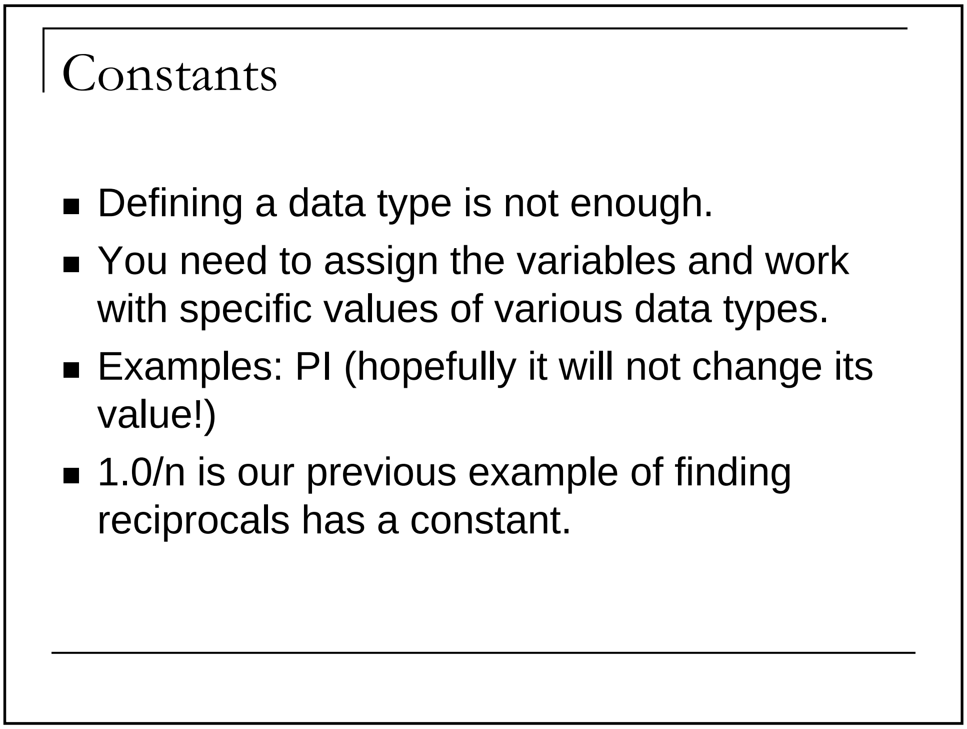
- A char data type is also an integer data type.
- If you want to interpret a char value as a character, you see the character it represents. If you want to view it as an integer, you see the ASCII value of that character.
- For example, the upper case A has an ASCII value of 65.
  - An eight-bit value representing the character A automatically represents the integer 65,
  - because to the computer A is recognized by its ASCII code, not by its shape, geometry or sound!



Pointer Data Type

- Pointers are addresses in memory.
- In order that the user can directly manipulate memory addresses, C provides an abstraction of addresses.
- The memory location where a data item resides can be accessed by a pointer to that particular data type. C uses the special character * to declare pointer data types.
- A pointer to a double data is of data type double *.
- A pointer to an unsigned long int data is of type unsigned long int *.
- A character pointer has the data type char *.

- We will study pointers more elaborately later in this course.



Constants

- Defining a data type is not enough.
- You need to assign the variables and work with specific values of various data types.
- Examples: PI (hopefully it will not change its value!)
- 1.0/n is our previous example of finding reciprocals has a constant.

Integer Constants

- An integer constant is a non-empty sequence of decimal numbers preceded optionally by a sign (+ or -).
- However, the common practice of using commas to separate groups of three (or five) digits is not allowed in C.
- Nor are spaces or any character other than numerals allowed.
- Here are some valid integer constants:

  332
  -3002
  +15
  -00001020304
- And here are some examples that C compilers do not accept:

3 332
2,334
-   456
2-34
12ab56cd

Hexadecimal values

- You can also express an integer in base 16, i.e., an integer in the **hexadecimal** (abbreviated **hex**) notation.
- In that case you must write either 0x or 0X before the integer. Hexadecimal representation requires 16 digits 0,1,...,15. In order to resolve ambiguities the digits 10,11,12,13,14,15 are respectively denoted by a,b,c,d,e,f (or by A,B,C,D,E,F). Here are some valid hexadecimal integer constants:

  0x12ab56cd -0X123456 0xABCD1234 +0XaBCd12

Real Constants

- Real constants can be specified by the usual notation comprising an optional sign, a decimal point and a sequence of digits. Like integers no other characters are allowed.
- Real numbers are sometimes written in the *scientific notation* (like 3.45x1067). The following expressions are valid for writing a real number in this fashion: 3.45e67 +3.45e67 -3.45e-67 .00345e-32 1e-15.
- You can also use E in place of e in this notation