Programming Languages

What we are going to discuss

1. ASCII, UNICODE
2. Writing “Hello World” in different programming languages
3. Low-level, High-level langauges
4. Programming errors
5. How to write a program

Decimal:

- Base-10, each number or **digit** takes on any one of 10 values: 0 to 9
- In general an *n-number* system takes on values *0 to (n-1)*
- Each ***place value*** increases by an order of 10 since it is base-10
- In general an *n-number* system place values increase by an order of *n*

Example - 5962; each place value increases by an order of 10

Working right to left from the least significant digit

2 is the 1's place or 10^0
6 is the 10's place or 10^1
9 is the 100's place or 10^2
5 is the 1000's place or 10^3

$5962 = (5 * 10^3) + (9 * 10^2) + (6 * 10^1) + (2 * 10^0) = 5000 + 900 + 60 + 2 = 5962$

Binary:

- Base-2, each **bit** takes on a value of 0 or 1
- Each ***place value*** increases by an order of 2 (power of 2)
- 8-bits are a **byte**
- 2,4, or 8 bytes are a **word**

ASCII and Unicode Characters

Unicode Characters

- Every character used in hardware is represented by a binary number
- Standardization required so going from one computer to next will interpret data the same
- **ASCII** (***American Standard Code for Information Interchange***) which is based on 7 bits (technically 2^0 to 2^6) or 2^7 = 128 characters
- Many languages going to **Unicode Worldwide Character Standard** - each character represented as 16 bits or 2^{16} = 65,536 characters
- First 128 Unicode characters are ASCII's 128 characters

Examples: A = 65
a = 97

ASCII TABLE

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	*[NULL]*
1	1	1	1	*[START OF HEADING]*
2	2	10	2	*[START OF TEXT]*
3	3	11	3	*[END OF TEXT]*
4	4	100	4	*[END OF TRANSMISSION]*
5	5	101	5	*[ENQUIRY]*
6	6	110	6	*[ACKNOWLEDGE]*
7	7	111	7	*[BELL]*
8	8	1000	10	*[BACKSPACE]*
9	9	1001	11	*[HORIZONTAL TAB]*
10	A	1010	12	*[LINE FEED]*
11	B	1011	13	*[VERTICAL TAB]*
12	C	1100	14	*[FORM FEED]*
13	D	1101	15	*[CARRIAGE RETURN]*
14	E	1110	16	*[SHIFT OUT]*
15	F	1111	17	*[SHIFT IN]*
16	10	10000	20	*[DATA LINK ESCAPE]*
17	11	10001	21	*[DEVICE CONTROL 1]*
18	12	10010	22	*[DEVICE CONTROL 2]*
19	13	10011	23	*[DEVICE CONTROL 3]*
20	14	10100	24	*[DEVICE CONTROL 4]*
21	15	10101	25	*[NEGATIVE ACKNOWLEDGE]*
22	16	10110	26	*[SYNCHRONOUS IDLE]*
23	17	10111	27	*[ENG OF TRANS. BLOCK]*
24	18	11000	30	*[CANCEL]*
25	19	11001	31	*[END OF MEDIUM]*
26	1A	11010	32	*[SUBSTITUTE]*
27	1B	11011	33	*[ESCAPE]*
28	1C	11100	34	*[FILE SEPARATOR]*
29	1D	11101	35	*[GROUP SEPARATOR]*
30	1E	11110	36	*[RECORD SEPARATOR]*
31	1F	11111	37	*[UNIT SEPARATOR]*
32	20	100000	40	*[SPACE]*
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	**0**
49	31	110001	61	**1**
50	32	110010	62	**2**
51	33	110011	63	**3**
52	34	110100	64	**4**
53	35	110101	65	**5**
54	36	110110	66	**6**
55	37	110111	67	**7**
56	38	111000	70	**8**
57	39	111001	71	**9**
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	**A**
66	42	1000010	102	**B**
67	43	1000011	103	**C**
68	44	1000100	104	**D**
69	45	1000101	105	**E**
70	46	1000110	106	**F**
71	47	1000111	107	**G**
72	48	1001000	110	**H**
73	49	1001001	111	**I**
74	4A	1001010	112	**J**
75	4B	1001011	113	**K**
76	4C	1001100	114	**L**
77	4D	1001101	115	**M**
78	4E	1001110	116	**N**
79	4F	1001111	117	**O**
80	50	1010000	120	**P**
81	51	1010001	121	**Q**
82	52	1010010	122	**R**
83	53	1010011	123	**S**
84	54	1010100	124	**T**
85	55	1010101	125	**U**
86	56	1010110	126	**V**
87	57	1010111	127	**W**
88	58	1011000	130	**X**
89	59	1011001	131	**Y**
90	5A	1011010	132	**Z**
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	**a**
98	62	1100010	142	**b**
99	63	1100011	143	**c**
100	64	1100100	144	**d**
101	65	1100101	145	**e**
102	66	1100110	146	**f**
103	67	1100111	147	**g**
104	68	1101000	150	**h**
105	69	1101001	151	**i**
106	6A	1101010	152	**j**
107	6B	1101011	153	**k**
108	6C	1101100	154	**l**
109	6D	1101101	155	**m**
110	6E	1101110	156	**n**
111	6F	1101111	157	**o**
112	70	1110000	160	**p**
113	71	1110001	161	**q**
114	72	1110010	162	**r**
115	73	1110011	163	**s**
116	74	1110100	164	**t**
117	75	1110101	165	**u**
118	76	1110110	166	**v**
119	77	1110111	167	**w**
120	78	1111000	170	**x**
121	79	1111001	171	**y**
122	7A	1111010	172	**z**
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	*[DEL]*

Programming Languages

Matlab:

```
disp('Hello World')
```

Python:

```
print(“Hello World”)
```

Ruby:

```
puts 'Hello World'
```

Java:

```
class HelloWorldApp {

public static void main(String[] args) {

System.out.println("Hello World!"); // Prints the string to the console. }

}
```

C:

```
#include <stdio.h>

int main(void) {
    printf("hello, world\n");
}
```

Assembly:

```
global _main
    extern _printf
    section .text
_main:
    push message
    call _printf
    add esp, 4
    ret
message:
    db 'Hello, World', 10, 0
```

Machine code:

```
b8    21 0a 00 00   #moving "!\n" into eax
a3    0c 10 00 06   #moving eax into first memory location
b8    6f 72 6c 64   #moving "orld" into eax
a3    08 10 00 06   #moving eax into next memory location
b8    6f 2c 20 57   #moving "o, W" into eax
a3    04 10 00 06   #moving eax into next memory location
b8    48 65 6c 6c   #moving "Hell" into eax
a3    00 10 00 06   #moving eax into next memory location
b9    00 10 00 06   #moving pointer to start of memory location into ecx
ba    10 00 00 00   #moving string size into edx
bb    01 00 00 00   #moving "stdout" number to ebx
b8    04 00 00 00   #moving "print out" syscall number to eax
cd    80            #calling the linux kernel to execute our print to stdout
b8    01 00 00 00   #moving "sys_exit" call number to eax
cd    80            #executing it via linux sys_call
```

Low Level: Machine Code (1st Generation)

0's and 1's
NOT portable from one computer architecture to the next
impossible to read

```
LOAD    100100
MULT    100110
STOR    100010
```

and the following variables:

```
rate    010001
hours   010010
wages   010011
```

Then, the following machine code calculates rate * hours

```
100100  (LOAD)    010001  (rate)
100110  (MULT)    010010  (hours)
100010  (STOR)    010011  (wages)
```

Low Level: Assembly (2nd Generation)

symbolic for CPU instructions
one line of code represents just one CPU instruction

```
LOAD   load
STOR   store
JMP    jump
```

NOT portable
An assembler translates assembly code to machine code

High Level

- Follow pre-defined syntax and rules of semantics
- Many high level languages portable among computer architectures
- Written in language easier to understand
- One line represents multiple lines of assembly

High Level: Compiled vs Interpreted

Compiled - Whole program read in and converted to machine language
Uses a compiler to convert and a linker to load and run/execute
Code run/executed AFTER compilation by a Linker

Source code → Object code
C, C++, FORTRAN

High Level: Compiled vs Interpreted

Interpreted - Program translated and executed line by line

Often called scripting languages, hence programs are called **scripts** or programs

Interactive environment

Can build on a program currently *running*, if you will, as long as it is still loaded in memory

Perl, MATLAB

High Level Languages: Generations

Third Generation
FORTRAN, BASIC, C first of the high level
Mostly portable between processors

Fourth Generation
Java, Ada, MATLAB, Mathematica
Completely portable
Graphics capabilities

Fifth Generation
GOAL - to use natural language
Requires speech recognition software

Programming Errors

Called **bugs**, hence debugging programs

Syntax errors will be identified during compilation or translation – language errors

Runtime errors – incorrect use of the language's library

Logic errors – fundamental problem in program logic

1. test all possible inputs or create boundary conditions

e.g.

If I am reading in height in inches, 69 inches is fine

but what about -50 inches

I probably want both a lower limit (anything less than 0 inches) and an upper limit (96 inches or 8 feet)

Approach to write a program

Take the Design Thinking approach:

1. **Empathize:** Think deeply about the problem.Understand the objectives and constraints.
2. **Define:** Define the problem in a formal way.
3. **Ideate:** Generate ideas to solve the problem.(write pseudocode)
4. **Prototype:** Develop prototype solutions.(write actual code)
5. **Test:**Make sure the solution actually works.Are there bugs? Can you crash the program? If the answer to any of the questions is yes, go back to previous steps.

In real world, each step can have significant consequences.

If your code is being used in a rocket to the mars, one bug can make the rocket explode. You don’t want that.

Syntax – set of rules

Algorithm – standard methods of processing; reduce complex task into smaller, simpler subtasks efficient!!

Good programs are:

1. readable
2. maintainable
3. reusable