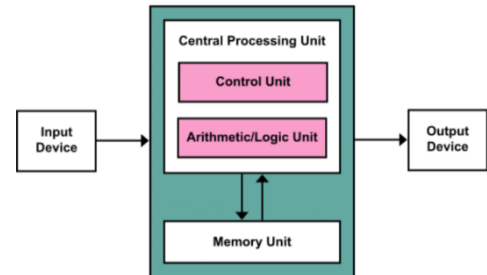


Engineering Computation Study Sheet

Computer: A device that carries out *programs* using arithmetic or logical expressions. This requires *hardware* and *software*.

Figure 1: CPU



- Input and output devices
- Memory
- CPU
- Secondary storage (long term)

8 bits = 1 byte

Machine language: Only the computer can understand, and all programs and data must be translated into *machine language* before the computer can understand it.

Assembler: Translates program into machine language

Compiler: Translates *high level program* into machine language

Build = preprocess + compile + link

- **Preprocess:** Executes preprocessor directives
- **Compile:** Checks *syntax* and translates code into *object code*
- **Link:** Links code with necessary library (e.g. <stdio.h>, <math.h>, etc.)

Run = load + execute

- **Load:** Loads executable code into the memory
- **Execute:** Computer performs operation based on the code

IDE (integrated development environment): A software has all tools previously mentioned (e.g. CodeBlocks)

Pseudo code: English statements that *describe* the desired operation of the computer.

Variables:

- A variable stores a value
- A variable is of a unique type (e.g. *int*, *double*, *float*, *char*, etc.)
 - o Type implies:
 - How much storage will be allocated
 - How it is encoded into bytes
- A variable will have an *address*

Byte = house && *memory* = street

Functions:

- Functions are used to *modularize* a program
- A function is a sub-program
- A function has 3 purposes:
 - o Take input
 - o Operate
 - o Output
- A function with a non-void return type must include a *return statement* (e.g. `return _____;`)
- Calling a function examples:
 - o `hello();`
 - o `prtmax(7, 9-6);`
 - o `x=max(5, 9);`

Figure 2: Function Declaration

```

aType functionName(parameterList)
{
    declarations;
    commands;
}
```

- `parameterList` contains the declaration of one or a list of input variables that will receive input data
 - Individual declarations are separated by “,” (commas).
 - the list can be empty (or made as `void`) if the function takes no input
- `functionName` is the name of the function
- `aType` is the type of the value returned by the function.
- If `aType` is `void`, it means that the function returns nothing
- Variable declaration and commands are inside “{ }”
- There is no “;” after }

Figure 3: Conversion Specifiers

Type	printf	scanf
short	%d, %i, %hd, %hi	%hd, %hi
int	%d, %i	%d, %i
long	%ld, %li	%ld, %li
unsigned short	%hu	%hu
unsigned int	%u	%u
unsigned long	%lu	%lu
float	%f, %e, %E, %g, %G	%f, %e, %E, %g, %G
double	%f, %e, %E, %g, %G	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG	%Lf, %Le, %LE, %Lg, %LG

Implicit rule of promotion:

- Float type values can be represented as a double type value without losing information
- E.g:
 - o float → double (works)
 - o double → float (doesn't)
- Low byte x High byte = High byte

Logical operators: “AND,” “OR,” and “NOT” are logical operators.

Function prototype:

- `type functionName (type1, type2, ...);`
- `type functionName (type1 var1, type2 var2, ...);`

Figure 4: Logical Operators

expression	C Operator	C expression
$x > y$	>	<code>x > y</code>
$x < y$	<	<code>x < y</code>
$x \geq y$	>=	<code>x >= y</code>
$x \leq y$	<=	<code>x <= y</code>
$x = y$	==	<code>x == y</code>
$x \neq y$!=	<code>x != y</code>
AND	&&	<code>x && y</code>
OR		<code>x y</code>
NOT	!	<code>!x</code>

The operator `!` is a unary operator.

Cast operator: can be used to apply a type to a variable (e.g. (aType) aValue). Cast *low-precision to high-precision* variables.

Figure 5: ASCII Chart

Char type:

```
myChar = 'a';
myChar = 97;
```

- Used to store characters
- The most common way to code characters into integers is through *ASCII code*

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	(NULL)	128	80	(SPACE)	160	A0	@	192	C0	à
1	1	(START OF HEADING)	129	81	!	161	A1	A	193	C1	á
2	2	(START OF TEXT)	130	82	"	162	A2	B	194	C2	â
3	3	(END OF TEXT)	131	83	#	163	A3	C	195	C3	ã
4	4	(END OF TRANSMISSION)	132	84	\$	164	A4	D	196	C4	ä
5	5	(ENQUIRE)	133	85	%	165	A5	E	197	C5	å
6	6	(ACKNOWLEDGE)	134	86	&	166	A6	F	198	C6	æ
7	7	(BELL)	135	87	'	167	A7	G	199	C7	ç
8	8	(BACKSPACE)	136	88	(168	A8	H	200	C8	è
9	9	(SHIFT/CONTROL TAB)	137	89)	169	A9	I	201	C9	é
10	A	(LINE FEED)	138	8A	*	170	AA	J	202	CA	ê
11	B	(VERTICAL TAB)	139	8B	+	171	AB	K	203	CB	ë
12	C	(FORM FEED)	140	8C	,	172	AC	L	204	CC	l
13	D	(CARRIAGE RETURN)	141	8D	-	173	AD	M	205	CD	í
14	E	(SHIFT/CONTROL SPACE)	142	8E	.	174	AE	N	206	CE	î
15	F	(SHIFT/IN)	143	8F	/	175	AF	O	207	CF	ó
16	10	(DATA LINK ESCAPE)	144	90	0	176	B0	P	208	D0	ô
17	11	(DEVICE CONTROL 1)	145	91	1	177	B1	Q	209	D1	õ
18	12	(DEVICE CONTROL 2)	146	92	2	178	B2	R	210	DA	ö
19	13	(DEVICE CONTROL 3)	147	93	3	179	B3	S	211	DB	ç
20	14	(DEVICE CONTROL 4)	148	94	4	180	B4	T	212	DC	à
21	15	(DEVICE CONTROL 5)	149	95	5	181	B5	U	213	DD	á
22	16	(DEVICE CONTROL 6)	150	96	6	182	B6	V	214	DE	â
23	17	(DEVICE CONTROL 7)	151	97	7	183	B7	W	215	DF	ã
24	18	(DEVICE CONTROL 8)	152	98	8	184	B8	X	216	EA	ä
25	19	(DEVICE CONTROL 9)	153	99	9	185	B9	Y	217	EB	å
26	1A	(DEVICE CONTROL 10)	154	9A	:	186	BA	Z	218	EC	æ
27	1B	(DEVICE CONTROL 11)	155	9B	;	187	BB	[219	ED	ç
28	1C	(DEVICE CONTROL 12)	156	9C	<	188	BC	\	220	EE	à
29	1D	(DEVICE CONTROL 13)	157	9D	=	189	BD]	221	EF	á
30	1E	(DEVICE CONTROL 14)	158	9E	>	190	BE	^	222	F0	â
31	1F	(DEVICE CONTROL 15)	159	9F	?	191	BF	_	223	F1	ã
32	20	(SPACE)	160	A0		192	C0		224	F2	ä
33	21	!	161	A1	!	193	C1	!	225	F3	å
34	22	"	162	A2	"	194	C2	"	226	F4	æ
35	23	#	163	A3	#	195	C3	#	227	F5	ç
36	24	\$	164	A4	\$	196	C4	\$	228	F6	è
37	25	%	165	A5	%	197	C5	%	229	F7	é
38	26	&	166	A6	&	198	C6	&	230	F8	ê
39	27	'	167	A7	'	199	C7	'	231	F9	ë
40	28	(168	A8	(200	C8	(232	FA	l
41	29)	169	A9)	201	C9)	233	FB	í
42	2A	*	170	AA	*	202	CA	*	234	FC	î
43	2B	+	171	AB	+	203	CB	+	235	FD	ó
44	2C	,	172	AC	,	204	CC	,	236	FE	ô
45	2D	-	173	AD	-	205	CD	-	237	FF	õ
46	2E	.	174	AE	.	206	CE	.	238		
47	2F	/	175	AF	/	207	CF	/	239		
48	30	0	176	B0	0	208	D0	0	240		
49	31	1	177	B1	1	209	D1	1	241		
50	32	2	178	B2	2	210	DA	2	242		
51	33	3	179	B3	3	211	DB	3	243		
52	34	4	180	B4	4	212	DC	4	244		
53	35	5	181	B5	5	213	DD	5	245		
54	36	6	182	B6	6	214	DE	6	246		
55	37	7	183	B7	7	215	DF	7	247		
56	38	8	184	B8	8	216	EA	8	248		
57	39	9	185	B9	9	217	EB	9	249		
58	3A	:	186	BA	:	218	EC	:	250		
59	3B	;	187	BB	;	219	ED	;	251		
60	3C	<	188	BC	<	220	EE	<	252		
61	3D	=	189	BD	=	221	EF	=	253		
62	3E	>	190	BE	>	222	F0	>	254		
63	3F	?	191	BF	?	223	F1	?	255		

Decision Structures:

- *If* statement
- *If else* statement
- *Switch* statement

Figure 7: If Statement with {}

```
if (S)
{
    Statement1;
    Statement2;
    ...
}
```

Figure 6: Nested else if

```
if (grade >= 90)
    printf("You have an A+\n");
else if (grade >= 85)
    printf("You have an A\n");
else if (grade >= 80)
    printf("You have an A-\n");
else if (grade >= 70)
    printf("You have a B\n");
...
else
    printf("You failed\n");
```

Figure 8: Switch Statement

```
switch (I)
{
    case CONST1:
        DoA;
        break;
    case CONST2:
        DoB;
        break;
    ...
    default:
        DoX;
}
```

- switch, case, default, and break are all keywords.
- I is an expression that evaluates to an integer.
- CONST1, CONST2 ... are integer literals.
- DoA, DoB, ... DoX are each a line of code or a block of code.
 - If any of them is a block of code, the block needs not to be enclosed by curly brackets { }.
- Arbitrarily many case's are allowed.

Loops:

- *For* loop
- *While* loop
 - o *Sentinel* is used to control the loop if it was to operate for an undefined # of loops
- *Do while* loop

```
for (initializeCounter; S; modifyCounter)
    DoX;
```

Figure 10: While loop

```
while (S)
    DoX;
```

Figure 11: Do While Loop

```
do
    DoX;
while(S);
```

Array:

- An *array* contains same type variables
- Type arrayName[arrayLength];
- float x[5]={0.0, 0.5, 1.0, 1.5, 2.0};
- Arrays can also be in *matrix* form e.g. type arrayName[numOfRows][numOfColumns];

Structures:

- Store many *different type* variables

Figure 12: Furthest Simplification of Structure Declaration

- Serves as new type: "struct structureName"
- To access a structures member:
 - o Use: *structureName.memberName*
- typedef can rename a type and makes structures simpler

```
typedef struct
{
    int ID;
    char name[200];
    float grade;
} REC;
```

Figure 13: Unary Operators '&' and '*'

Highlight ("&" and "*" as a Reciprocal Pair of Operators)

- &X gives the address for variable X.
- *X gives the "variable" having address X.

Highlight (A New View of Variable)

A variable is nothing more than a block of bytes in the memory, which has been associated with a prescribed method specifying how it should be interpreted as a number or as numbers.

using *

Pointers:

- Pointers are used as variables that store addresses
- Declared using:
 - o varType * pointerName = &var
- NULL pointers point nowhere and NULL == 0 by default
- A pointer can be dereferenced

Figure 14: Function Using Array Pointers to Receive Array Input

```
returnType myFunc(someType* ptrArray, int lengthArray)
{
    ...
}
```