CH. 9

W5

■ **STRINGS & CHARACTERS** – "Enumerated Data Types"

! "A data type defined by listing all of its elements is an **ENUMERATED** data type."

• **EX:**
```
#define   Sunday     0
...
#define   Saturday   6
```

• **General definition:**

```
typedef   enum
{ Sunday [=0], Monday, ..., Saturday
} weekdayT;
                    /* definition of type "weekdayT"   */
                    /* numbered implicitly from 0 to 6 */
```

➔ **Can now declare:** `weekdayT   weekday;`

OR:
```
typedef   enum                     |  typedef   enum
{ Sun=0, Mon=1, ..., Sat=6         |  { Sun=0, Mon, Tue, ..., Sat
} weekdayT;                        |  } weekdayT;
                                                  /* Implicitly        */
                                                  /* incrementing by 1 */
```

• **EX:** Type "bool":

```
typedef   enum
{ FALSE [=0], TRUE [=1]
} bool;
```

- "**Characters**" are enumerated!

→ Every **char** has a unique number assigned to it.

ASCII = **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

→ **P. 309**:

	0	1	2	3	4	5	6	7	8	9
0										
10	\n									
20										
30			space							
40									0	1
50	2	3	4	5	6	7	8	9		
60						A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z							a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z						▨	▨

→ "**Code**" of character 'A' is 65.

→ **128** character codes — "1-byte codes"

- **SPECIAL**:

\\	char '\'
\b	backspace
\n	newline
\0	**null** character
\"	char '"'

"escape sequences"

■ Operations for characters: **CHARACTER ARITHMETIC**

(i) char + int '0'+5 = 48+5 = 53 = code for '5'

(ii) char - int 'Z'-2 = 90-2 = 88 = code for 'X'

(iii) char - char 'a'-'A' = 97-65 = 32 = "distance of codes"

(iv) comparison c1<c2 is TRUE if code for c1 is smaller than code for c2

(v) digit/char '5' mapped to int 5: ch - '0'

• EX: '7'-'0' = 55-48 = 7

(vi) if (ch>='0' && ch<='9') then "ch is a digit"
if (ch>='A' && ch<='Z') then "ch is a cap. letter"

(vii) MORE operations in <ctype.h>:

RETURN TRUE/FALSE
- islower (ch)
- isupper (ch)
- isalpha (ch) /* true for letters */
- isdigit (ch)
- isalnum (ch)
- ispunct (ch) /* punctuation characters */
- isspace (ch) /* '␣', '\t', '\n', '\f', ... */ ↑ form feed

RETURN CHAR
- tolower (ch)
- toupper (ch)

(viii) "Tricks":

```
                65         90
• for (ch = 'A'; ch <= 'Z'; ch++)

• bool IsVowel (char c)
  { switch (toLower(ch))
     { case 'a': case 'e': case 'i': case 'o': case 'u':
                 return (TRUE);
       default: return (FALSE);
     }
  }

  main ()
  { char ch;
    printf ("Vowels are:" );
     for (ch = 'A'; ch <= 'Z'; ch++)
     { if (IsVowel (ch)) printf (" %c", ch);
     }
     printf ("\n");
  }
```

■ STRINGS

increasing abstraction ↑

Roberts: strlib.h

ANSI C: string.h

Language-level (C) operations

machine-level operations

↓ decreasing abstraction

→ lower-level detail operations

String is an ABSTRACT data type, defined by the operations for it —

● NOT its representation!

W5

The strlib.h interface:

- **Length:** string str; ... str = GetLine(); ...
  ➡ StringLength(str) = no. of characters

- **Selecting ith char:** string

```
 0 1 2 3 → INDEX
|H|E|L|L|O|␣|W|O|R|L|D|.| ...        |(\0)
```

```
char c; string str;
str = "HELLO␣WORLD.";
c = IthChar(str, 4); ➡ c=='O'
```

- **Concatenation:** Concat("Hello␣", "World.");
  ➡ returns string "Hello␣World."

- **Multiple concatenation:**

```
String ConcatNTimes(int n, string str)
{ string res;
  int i;
  res = ""     /* empty string */
  for (i = 0; i < n; i++)
  {  res = Concat(res, str);
  }
  return(res);
}
```

➡ ConcatNTimes(3, "ABBA")
   returns "ABBAABBAABBA"

• **Conversion:** from character to string "**CharToString**"

→ 'A' becomes "A"

/* difference: **NULL** character */
/* appears at end of string! */

• **Reverse string:**

```
string ReverseString (string str)
{ string res;
  int    i;
  res  = "";
  for (i=0; i< StringLength (str); i++)
  { res = Concat ( CharToString (IthChar (str, i)),
                   res);
  }
  return(res);
}
```

/*concatenated in this order */

• **EX:** str : "abc"

→

i	res
0	"a"
1	"ba"
2	"cba"

• **Sub-string:** str = SubString ("Hello!", 1, 3);

→ returns "ell"

W5

• **Comparison:** EX: aab > Aab

abc < abcd

$$\text{StringCompare(str1, str2)} \begin{cases} <0, & \text{str1} < \text{str2} \\ ==0, & \text{str1} == \text{str2} \\ >0, & \text{str1} > \text{str2} \end{cases}$$

/* standard lexicographical order */

• **Searching:** ➡ Searching for a character or a (sub-) string

• EX: • FindChar('l', "Hello!", 1);
returns 2 ↑ where to start search

• FindChar('a', "Hello!", 0);
returns -1 /* not found */

• FindString ("World", "Hello␣World!", 0);
returns 6

• **Case Conversion:** • word = ConvertToUpperCase ("Hello!\n");
returns "HELLO!\n"

• **Numeric Conversion:** EX: • IntegerToString (123) ➡ "123"

• RealToString (3.14) ➡ "3.14"

• StringToInteger ("42") ➡ 42

• StringToReal ("3.1415") ➡ 3.1415

➡ TAB. 9-3: ALL strlib.h FUNCTIONS!