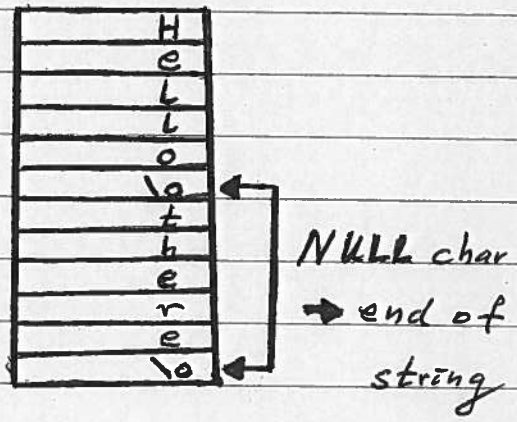
CH. 14

More on Strings

String = array of char



- Ex.: #define word1 "Hello"
  #define word2 "there"

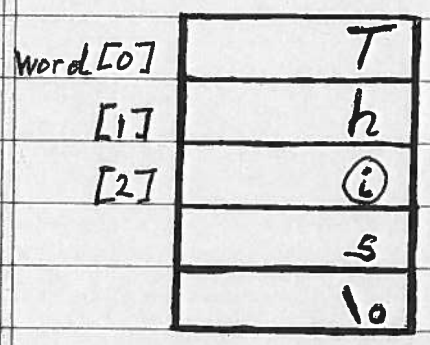
- Ex.: Find 1st Vowel:

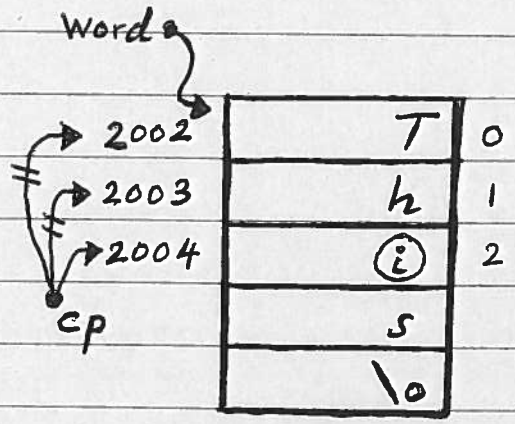
(i) array-based

```
int Find1stVowel (char word[])
{ int  i;
  for (i=0; word[i]!='\0'; i++)
  { if (IsVowel(word[i]))
        return (i);
  }
  return(-1);
}
```

(ii) pointer-based

```
int Find1st Vowel
      ( char *word)
{ char  *cp;
  for (cp=word;
        *cp!='\0'; cp++)
  { if (IsVowel (*cp))
  }     return (cp-word);
}return(-1);
```



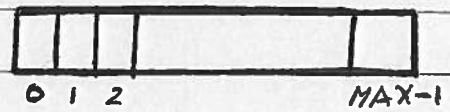


cp - word == 2

- **Array-based vs. pointer-based implementation:**

(i) #define MAX 1000000

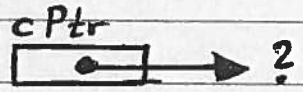
```
char    cArr [MAX];
```



effective size: n

→ waste of space when $n \ll MAX$

(ii) char *cPtr; /* no memory needed here */



Later in program: **DYNAMIC ALLOCATION**

```
cPtr = GetBlock (blocksize);
```

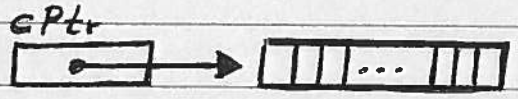
① ② ↑ in genlib.h

① equivalent to:

```
cPtr = malloc (blocksize);
if (cPtr == NULL) Error("...");
```

② blocksize is a variable:

```
int blocksize;
blocksize = ...;  /* set during execution */
```

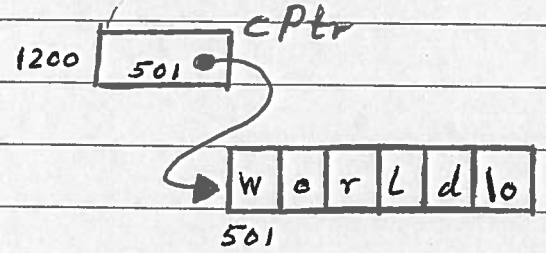


# of bytes == blocksize

- Pointers - Arrays - Strings:

  "Strings = Arrays of characters"

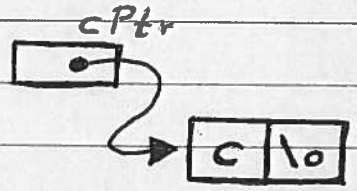
```
char *cPtr;
cPtr = "world";
```



→ STRING = POINTER TO ARR. OF CHARS.

- Ex:

```
string CharToString (char c)
{
    char *cPtr;

    cPtr = GetBlock (2);
    cPtr[0] = c;
    cPtr[1] = '\0';
    return (cPtr);    /* returns pointer to array */
}
```



- Note: - ANSI C : string.h
  → programmer must worry about mem. allocation

  - Roberts : strlib.h
  → takes care of memory allocation

- **Strings:** copying and concatenating

→ void strcpy (string dest, string source);
/* copies source into dest */

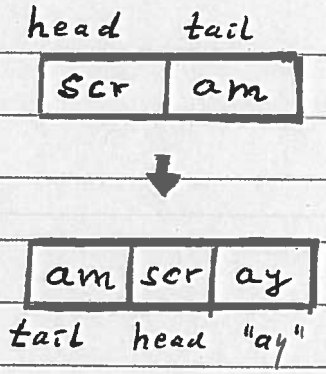
e.g.: strcpy (buffer, line);

→ void strncpy (string dest, string source, int n);
/* copies at most n chars */

→ void strcat (string dest, string source, int n);
(n)

/* appends (at most n) source */
/* (chars) to destination */

- **Ex:**




```
string head, tail;
char pigWord [MAX+1];

head = "scr";
tail = "am";
```

strcpy (pigWord, tail); | a | m | \0 |

strcat (pigWord, head); | a | m | s | c | r | \0 |

strcat (pigWord, "ay"); | a | m | s | c | r | a | y | \0 |

• **Ex.:** invert.c

→ Prince ↦ Prince

Bernd Hamann ↦ Hamann, Bernd

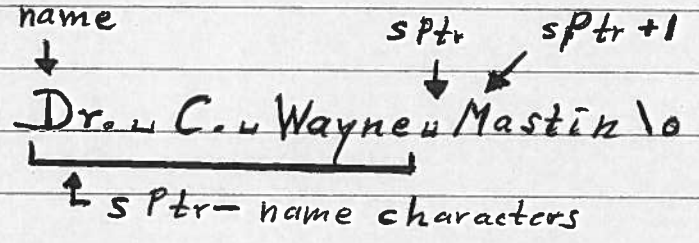
Dr. C. Wayne Mastin ↦ Mastin, Dr. C. Wayne

→ "Find last '␣' and use string functions."

```
...
#define MAX 40
...
void InvertName (char res[], char name[]);
...
main()
{ char *name;
  char invName [MAX+1];
  ...
  while (TRUE)
  { printf ("Name: ");
    name= GetLine();
    if (strlen (name==0)) break;    /* STOP */
    InvertName (invName, name);
    printf ("Inverted name: %s \n", invName);
  }
}
...
```

...

```
void InvertName(char res[], char name[])
{
  int Len;
  char *sPtr;                    /* to indicate last '␣' */

  Len = strlen(name);
  sPtr = strrchr(name, '␣');     /* ptr to last '␣' */
  if (sPtr != NULL) Len++;       /* for ',' character */
  if (Len > MAX) Error("too long");
  if (sPtr == NULL)              /* no '␣' */
  { strcpy(res, name);           /* Prince → Prince */
  }
  else
  { strcpy(res, sPtr+1);         ①
    strcat(res, ",␣");           ②
    strncat(res, name, sPtr-name); ③
    res[Len] = '\0';             ④
  }
}
```



① Mastin

② Mastin,␣

③ Mastin,␣Dr.␣C.␣Wayne

④ Mastin,␣Dr.␣C.␣Wayne\0

CH. 16

FILES – "2D Array Structures"

→ column

↓ row

T	h	i	s	␣	i	s	\n				
a	\n										
f	i	l	e	(	t	x	t	)	\n		

stored on disk (not main memory) – "secondary memory"

+ end-of-file symbol: **eof / EOF**

- **Operations:** – opening / closing a file
  – reading from / writing to a file
  "input/output (I/O) operations"

- stdio.h: interface for file I/O
- **Declaration:** FILE *infile;

- **Opening file:**

```
file_ptr_var
        = fopen(file_name, mode);
```

  – file_name, mode: of type string
  – mode: "r"-read / "w"-write / "a"-append

Ex:

```
infile = fopen("input.txt", "r");
if (infile == NULL)
    Error("Cannot open file");
```

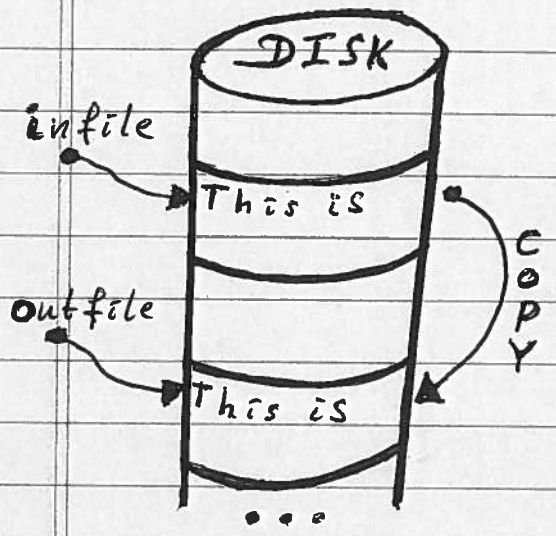
- **Closing file:**

```
fclose(file_ptr_var);
```

Ex: fclose(infile);

- I/O commands:
  - Read/write char-by-char: getc/putc
  - " Line-by-line: fgets/fputs
  - " formatted data: fscanf/fprintf

- **Ex:** "Copying a file char-by-char"



- prototype definitions:

```
void CopyFile
    (FILE *infile,
     FILE *outfile);

FILE *OpenFile
    (string prompt,
     string mode);
```

...

①
```
main()
{ FILE *infile, *outfile;
  printf ("Copying a file... \n");
  infile = OpenFile ("File to be copied:␣", "r");
  outfile = OpenFile ("Name of copy:␣", "w");
  CopyFile (infile, outfile);
  fclose (infile);
  fclose (outfile);
}
```

2

```
FILE *OpenFile (string prompt, string mode)
{ string filename;
  FILE *res;

  while (TRUE)        /* try MULTIPLE times: cannot */
  {                   /* open file for output if it's */
                      /* opened for input...  */
    printf ("%s", prompt);
    filename = GetLine ();
    res = fopen (filename, mode);
    if (res != NULL) break;
    printf ("Cannot open file \"%s\" \n", filename);
  }
  return (res);
}
```

try multiple times

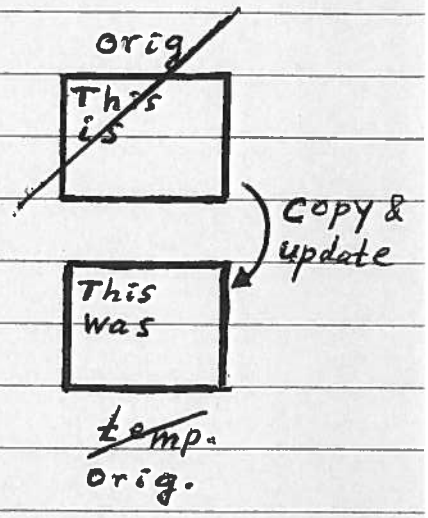
3

```
void CopyFile (FILE *infile, FILE *outfile)

{ int ch;

                  returns int: for EOF      EOF == -1
  while ( (ch = getc (infile)) != EOF )
  {  putc (ch, outfile);
  }

}
```

- **Updating**

1. Open original file for input.
2. Open temp. file for output.
3. Copy original file to temp. file; perform all updates.
4. Close original and temp. files.
5. Delete original file.
6. Rename temp. file.

WHY? "Illegal" to open file for output if it's open for input.../Cannot write to the (same) file I am reading from...



- **Ex:** Converting file from lower-case to upper-case

```
...
void UpperCaseCopy(FILE *infile, FILE *outfile);
...
main()
{ string filename, tmp;
  FILE *infile, *outfile;

  printf("Converting from lower-case to upper-case\n");
  ...
```

```
...
                                        /* Open orig. file */
while (TRUE)
{ printf("File name: ");
  filename = GetLine();
1.  infile = fopen(filename, "r");
    if (infile != NULL) break;
    printf("Cannot open file %s.\n", filename);
}

    tmp = tmpnam(NULL);
                                        /* tmpnam defined in stdio.h */
2.                                      /* returns string as temp. name */
    outfile = fopen(tmp, "w");
    if (outfile == NULL)
        Error("Cannot open temp. file.");

3.  UpperCaseCopy(infile, outfile); ➡...

4.  fclose(infile);
    fclose(outfile);

5.,6. if (    (remove(filename) != 0)
          || (rename(tmp, filename) != 0))
        Error("Unable to remove orig. file
                  and/or rename temp. file");
}
```

- **Ex:** Line-oriented I/O (fgets, fputs)

"Copy line-by-line":

...

```
void CopyFile (FILE *infile, FILE *outfile)
{
  char buffer [MAX];

                          array of char        EOF
  while ( fgets (buffer, MAX, infile) != NULL )
  {
    fputs ( buffer , outfile );
  }
}
```

string

- **Formatted I/O**

write	read
printf → to std. output	scanf → from std. input
fprintf → to file	fscanf → from file
sprintf → to char array	sscanf → from string / char array

- **Reading strings:**

→ char word[MAX];
fscanf (infile, "%s", word);

→ BETTER: fscanf(infile, "%24s", word);
length limit

• **Ex:** Table of "Elements" (15-4 in textbook)

File:

```
Hydrogen, H, 1, 1.008 \n
Helium, He, 2, 4.003 \n
...
Neon, Ne, 10, 20.183 \n
EOF
```

Desired table (on standard output):

1.	Hydrogen (H)	1.008
2.	Helium (He)	4.003
...	...	
10.	Neon (Ne)	20.183

→ **Reading file:**

```
nscan = fscanf (infile,
                 "%15[^,], %2[^,], %d, %Lf%c",
                 elName, elSym, &atNum, &atWei, &termch);
```

nscan = no. of successful conversions → must be 5

read string (≤15 chars.) up to first ','

elName, elSym: char arrays – treated as addresses!

termch: must be '\n'

where:

```
char elName [MAX+1];
char elSym [MAX_SYM+1];
int atNum;
double atWei;
char termch;
int nscan;
```

→ **Printing** table (on standard output / to a string):

First: `sprintf (nameBuf, "%s␣(%s)", elName, elSym);`

↑ print to a string

Then: `printf ("%3d.␣%-20s␣%8.3f\n",`

↑ print to standard output

`atNum, nameBuf, atWei);`

Where: `char nameBuf[MAX+MAX_SYM+4];`

`/* +4: for '␣', '(', ')', '\0' */`

→ See Roberts textbook for detailed program.