Strings (cont'd)

CS2023 Winter 2004

Outcomes: Strings (part 2)

- “C for Java Programmers”, Chapter 9
- Other textbooks on C on reserve
- After the conclusion of this section you should be able to
  - Use functions from string.h that manipulate strings
  - Process arguments to main

Line-Oriented String I/O

One character at a time:

```
int read_line(char *str, int n){
  int ch;
  int i = 0;

  while ((ch = getchar()) != '\n' && ch !=
      EOF)
    if(i < n)
      str[i++] = ch;

  str[i] = '\0' /* terminate string */
  return i; /*number of characters stored*/
}
```

Line-Oriented String I/O

`char* gets(char *buf);`

- like **`fgets()`** but if end-of-line has been encountered, it is *not* stored in **`buf`**
- reading does not stop until end-of-line encountered, so can lead to buffer overflow and can be maliciously exploited!

NEVER USE THIS FUNCTION!

C String Operations

To compute the length of a string, use:

```
size_t strlen(const char *string);
```

- Doesn't compute length of block pointed to by string, but the length of the string up to but not including first `'\0'`
- Following is always true!

```
if (strlen(x) - strlen(y) >= 0)
```

- since **`strlen`** returns unsigned type. Use instead:

```
if (strlen(x) >= strlen(y))
```

C String Operations

- Can't directly compare or copy strings:

```
char *s1, *s2;
s1 = "abc"
s2 = s1 /* does not copy string!/
if (s1 == s2) ...
```

- Use C string library (**`string.h`**)
- C string library expects initialized (null-terminated) strings

C String Operations

To copy **`src`** to **`dest`** and return **`dest`**:

```
char *strcpy(char *dest, const char *src);
```

- **`strcpy`** has no way to check that string pointed to by **`src`** will actually fit into the block pointed to by **`dest`**
- **`strcpy`** copies **`src`** up to the first null character
- return value usually discarded, unless part of longer expression:

  ```
  strcpy(str2, strcpy(str1, "abcd"));
  ```

  – both **`str1`** and **`str2`** now contain "**`abcd`**"

C String Operations

Append (or, "catenate") **`src`** to **`dest`** and return **`dest`**:

```
char *strcat(char *dest, const char *src);
```

```
#define SIZE 4
char *dest;

if((dest
  =calloc(sizeof(char)*(SIZE+1)))== NULL)
    error
strcpy(dest, "Hello");
dest[0] = '\0';
strcat(dest, "Hi"); /*dest points to
  "Hi"*/
strcat(dest, " how");/*too long for
  dest*/
```

```
char *strdup(const char *s) {
/* return a copy of s */
   char *kopy;    /* copy of s */

   if((kopy = calloc(strlen(s) + 1, sizeof(char)))
             == NULL)
      return NULL;
   strcpy(kopy, s);

   return kopy;
}
```

C String Operations: Comparisons

- To lexicographically compare `s1` and `s2`:

```
int strcmp(const char *s1, const char
  *s2);
```

  - returns a negative value if `s1` is less than `s2`, 0 if the two are equal, a positive value of `s1` is greater than `s2`.

- To lexicographically compare `n` characters `s1` and `s2`:

```
int strncmp(const char *s1, const char
  *s2, size_t n);
```

C String Operations

- Comparison rules:
  - All upper case letters less than lower-case letters
  - Digits are less than letters
  - Spaces are less than all printing characters
- String search functions
  - **`strchr, strrchr, strstr, strspn, strcspn, strpbrk`**
  - C for Java Programmers, pp. 327-336

Implementation of strcat

```
char *strcat(char *s1, const char *s2) {
  char *p;
  p = s1;
  while (*p != '\0')
    p++;
  while (*s2 != '\0') {
    *p = *s2;
    p++;
    s2++;
  }
  *p = '\0'
  return s1;
}
```

Implementation of strcat (condensed)

```
char *strcat(char *s1, const char *s2) {
  char *p = s1;

  while (*p)
    p++;
  while (*p++ = *s2++)
    ;
  return s1;
}
```

"string copy" idiom

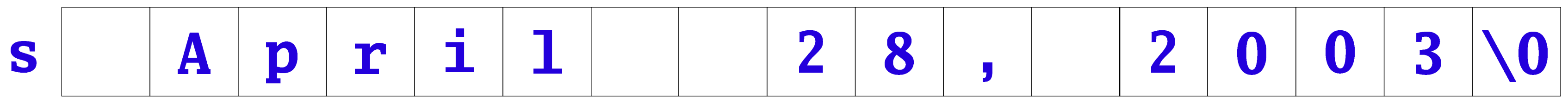
Processing Tokens

```
char *strtok(char *str, const char *sep);
```

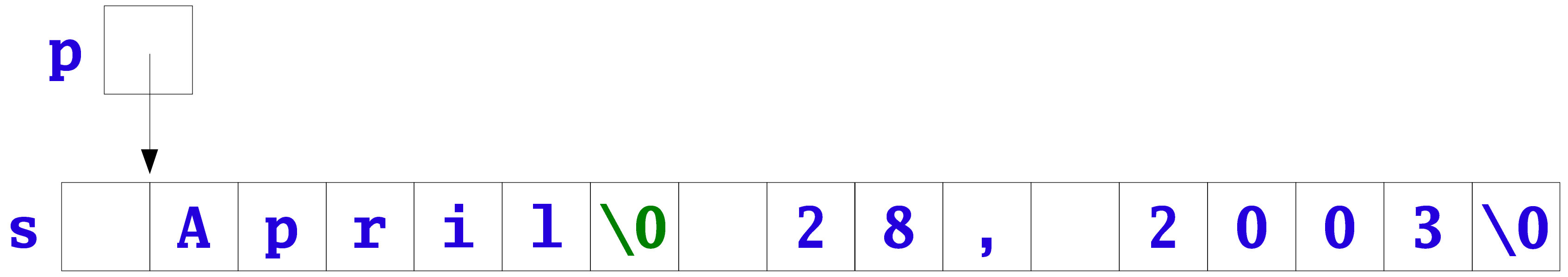
- separates **`str`** into tokens, using characters from **`sep`** as separators.
  - The first parameter **`str`** may be **`NULL`** (but not in the first call).
  - The *first* call takes the non-null first parameter, and returns a pointer to the first token (skipping over all separators)
  - All *subsequent* calls take **`NULL`** as the first parameter and return a pointer to the next token.
  - If the first call does not find any characters in **`sep`**, the function returns **`NULL`**.
  - Modifies the string being tokenized (to preserve a string, you have to make a copy of it before you call **`strtok()`**).

strtok example

- Extract month, day, and year from date written in form: *month day*, *year* (spaces or tabs)

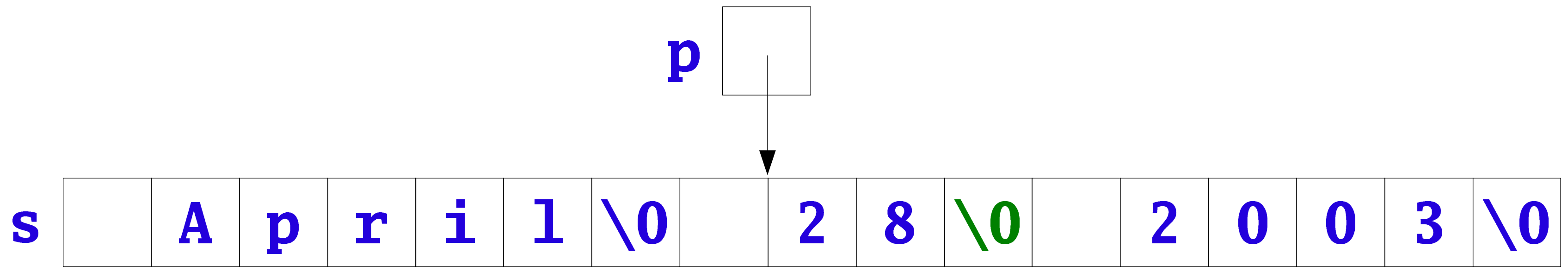


```
p = strtok(s, " \t");
```

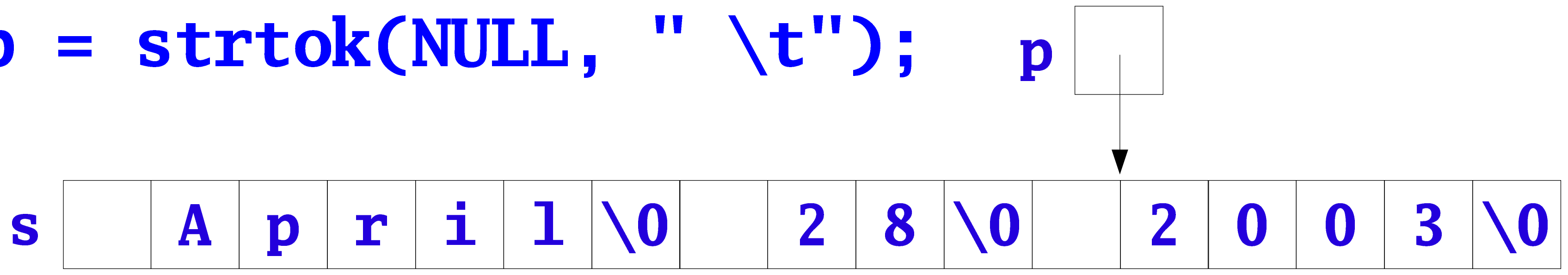


strtok example

```
p = strtok(NULL, " \t,");
```



```
p = strtok(NULL, " \t");
```



String to Number Conversions

```
double strtod(const char *s, char **p);
long strtol(const char *s, char **p, int
              base);
unsigned long strtoul(const char *s,
                          char **p, int base);
```

- Convert a string **`s`** *to* a number. If the conversion failed: **`*p`** is set to the value of the original string **`s`**
- Otherwise, **`p`** is set to point to the first character in the string **`s`** immediately following the converted part of this string.
  - A default **`base`**, signified by 0, is decimal, hexadecimal or octal, and it is derived from the string. (It also may be any number from 2 to 36).

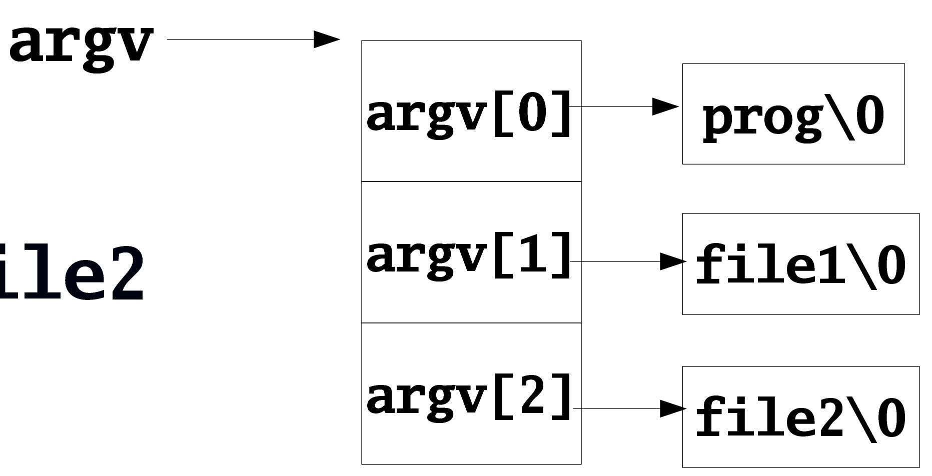
String to Number Conversions

```
double atof(const char *s);
int atoi(const char *s);
int atol(const char *s);
```

For compatibility with older versions of C.

Main Function's Arguments

prog file1 file2



```
int main(int argc, char **argv);
int main(int argc, char *argv[]);
```

Main Function's Arguments

```
int main(int argc, char **argv) {
…
   switch(argc) {
   case …
   default: fprintf(stderr, "usage: %s …
  \n", argv[0]);
            return EXIT_FAILURE;
}
```

This idiom only checks the number of required arguments, not their types or values

Main Function's Arguments

To pass numerical values on the command line; for example:

in a program, which displays *up to* the first **`n`** lines from a file:

```
    show -n fname
```

This program can be invoked without the first argument (**`-n`**), to display *up to* the first 10 lines.

Assuming we have:

```
int display(const char *fname, int n,
               int Max);
```

```
#define DEFAULT 10
#define MAX     80
int main(int argc, char **argv) {
  int lines = DEFAULT;
  switch(argc) {
  case 3: /* retrieve the number of lines argument */
   if(argv[1][0] != '-' ||
       sscanf(argv[1] + 1, "%d", &lines)!=1 || lines <= 0)

     return EXIT_FAILURE;
   argv++;          /* no break: retrieve filename */
  case 2:  if(display(argv[1], lines, MAX) == 0)
              return EXIT_FAILURE;
           break;
  default:
      return EXIT_FAILURE;
 }
 return EXIT_SUCCESS;
}
```

Main Function's Arguments

Redirection is not a part of the command line of a program.

```
program one two < f1 > f2
```

has two command line arguments, not six.

```
int main(int argc, char **argv){
  char line[MAX], *p;
  FILE *f;
  f = stdin;
  switch(argc) {
  case 2:
    f = fopen(argv[1], "r");
  case 1:
    break;
  default:
    fprintf(stderr, "usage: %s or %s file\n",
            argv[0], argv[0]);
    return 0;
  }
  while (fgets(line, MAX, f) != NULL) {
    p = strtok(line, " \t");
    do printf("%s\n", p);
     while((p = strtok(NULL, " \t")) != NULL);
  }
  return 0;
}
```