Notes 13.0: String and memory functions

COMP9021 Principles of Programming

School of Computer Science and Engineering
The University of New South Wales

2014 session 1

The string.h header file

Recall that `\0` is the `NUL` character, and that strings are arrays of characters distinct to `\0` that end in `\0`.

The contents of a string denotes the sequence of all characters that make up the array except for the final `\0`.

Distinguish between a null character pointer (`NULL`) and a nonnull pointer to an empty string (an array of length 1 with `\0` as single element).

The `string.h` header file provides the prototypes of functions from the standard library that analyze and manipulate strings and blocks of memory, the latter differing from the former in that the `NUL` character plays no particular role, either not occurring at all or occurring many times in the block.

Types of functions and types of parameters. The strlen() function

Function parameters and types of functions that refer to lengths of strings and blocks of memory are of type `size_t` (usually `unsigned long`).

- Function parameters and types of functions that refer to strings are of type `char *` or `const char *`, depending on whether the strings can be modified.
- Function parameters and types of functions that refer to blocks of memory are of type `void *` or `const void *`, depending on whether the blocks of memory can be modified.

```
size_t strlen(const char *s)
```

returns the number of characters in the contents of the string located at `s`.

This function is illustrated in length.c.

The strcat() and strncat() functions

```
char *strcat(char *start, const char *end)
```

appends the string located at `end` to the contents of the string located at `start`, at the location where the latter ends, and returns the (unchanged) value of `start`. The allocated memory segment referred to by `start` is supposed to be large enough to be able to store as many extra characters as in the contents of the string located at `end`.

```
char *strncat(char *start, const char *end, size_t n)
```

behaves as `strcat()` with the contents of the string located at `end` reduced to its initial segment of length `n` if longer.

These functions are illustrated in concatenate.c.

The strcmp(), strncmp(), and memcmp() functions

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

compares the strings located at `s1` and `s2`, returning a value smaller than `0` if the former is lexicographically smaller than the latter, `0` if both are identical, and a value greater than `0` otherwise.

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

behaves as `strcmp()` with the contents of the strings `s1` and `s2` reduced to their initial segment of length `n` if longer, respectively.

```
int memcmp(const void *pt1, const void *pt2, size_t n)
```

compares the sequence of `n` characters located at `s1` and `s2`, returning a value smaller than `0` if the former is lexicographically smaller than the latter, `0` if both are identical, and a value greater than `0` otherwise.

These functions are illustrated in compare.c.

The strcpy(), strncpy(), memcpy() and memmove() functions (1)

```
char *strcpy(char *dest, const char *source)
```

copies the string located at `source` to location `dest`, and returns the (unchanged) value of `dest`. The allocated memory segment referred to by `dest` is supposed to be large enough to be able to store the string located at `source`.

```
char *strncpy(char *dest, const char *source, size_t n)
```

copies exactly `n` characters at location `dest`, namely, the contents of the string *s* located at `source` reduced to its initial segment of length `n` if greater, followed by *p* `NUL` characters if the contents of *s* is of length `n` minus *p*, and returns the (unchanged) value of `dest`.

The behaviour of these functions is undefined if the strings overlap in memory.

The strcpy(), strncpy(), memcpy() and memmove() functions (2)

```
void *memcpy(void *dest, const void *source, size_t n)
```

and

```
void *memmove(void *dest, const void *source, size_t n)
```

copy at location `dest` the `n` characters stored at location `source`, and return the (unchanged) value of `dest`.

The behaviour of `memcpy()` is undefined if the blocks of memory overlap, whereas `memmove()` is safe in this case and behaves as if the `n` characters stored at `source` were first copied to a disjoint area and then copied at location `dest`.

These functions are illustrated in copy.c.

The strchr(), strrchr() and memchr() functions

```
char *strchr(const char *s, int c)
```

returns the location of the first occurrence of the character of code `c` in the string located at `s`, if such a character exists, and `NULL` otherwise.

```
char *strrchr(const char *s, int c)
```

returns the location of the last occurrence of the character of code `c` in the string located at `s`, if such a character exists, and `NULL` otherwise.

```
void *memchr(const void *pt, int v, size_t n)
```

returns the location of the first occurrence of the value `v` in the first `n` characters stored at location `pt`, if such a character exists, and `NULL` otherwise.

These functions are illustrated in find_character.c.

The strspn(), strcspn() and strpbrk() functions

```
size_t strspn(const char *s, const char *set)
```

returns the length of the longest initial segment of the string located at `s` consisting of characters that all occur in the contents of the string located at `set`.

```
size_t strcspn(const char *s, const char *set)
```

returns the length of the longest initial segment of the string located at `s` consisting of characters none of which occurs in the contents of the string located at `set`.

```
char *strpbrk(const char *s, const char *set)
```

returns the location of the first occurrence in the string located at `s` of a character in the contents of the string located at `set`, if such a character exists, and `NULL` otherwise.

These functions are illustrated in character_set.c.

The strstr() function

```
char *strstr(const char *start, const char *inside)
```

returns the location in the string located at `start` of the leftmost occurrence of the contents of the string located at `inside`, if such a string exists, and `NULL` otherwise.

This function is illustrated in substring.c.

The strtok() function (1)

```
char *strtok(char *s, const char *set)
```

is used to extract from `s` tokens separated by characters from `set`.

- It is called with the first argument set to `s` to extract the first token and then to `NULL` to extract the subsequent tokens.
- Between successive calls, the second argument can differ to change the token separators.

Suppose that `s` is not `NULL`. Then an internal state pointer is set to the value of `s`, and execution continues as if `s` had been `NULL`.

The strtok() function (2)

Suppose that `s` is `NULL`. If the internal state pointer is `NULL` then it remains equal to `NULL` and is returned by `strtok()`. Suppose that the internal state pointer is not `NULL`.

- If all characters in the string located at the value of the internal state pointer occur in the string located at `set` then `strtok()` returns `NULL` and the internal state pointer is set to `NULL`.
- Otherwise, let p be the location, in the string located at the value of the internal state pointer, of the leftmost character that does not occur in the string located at `set`. The string located at p contains a first occurrence o of a character c that either occurs in the string located at `set`, in which case o is overwritten with `\0`, or is equal to `\0`. Then the internal state pointer is set to the location of o if c is not `\0`, and to the next location otherwise. Finally, p is returned.

This function is illustrated in tokens.c.