17.2 Dynamically Allocated Strings

Dynamic storage allocation is often useful for working with strings. Strings are stored in character arrays, and it can be hard to anticipate how long these arrays need to be. By allocating strings dynamically, we can postpone the decision until the program is running.

Using `malloc` to Allocate Memory for a String

The `malloc` function has the following prototype:

```
void *malloc(size_t size);
```

`malloc` allocates a block of `size` bytes and returns a pointer to it. Note that `size` has type `size_t`, an unsigned integer type defined in the C library. Unless we're allocating a very large block of memory, we can just think of `size` as an ordinary integer.

`size_t` type ➤7.6

Using `malloc` to allocate memory for a string is easy, because C guarantees that a `char` value requires exactly one byte of storage (`sizeof(char)` is 1, in other words). To allocate space for a string of `n` characters, we'd write

```
p = malloc(n + 1);
```

where `p` is a `char *` variable. (The argument is `n + 1` rather than `n` to allow room for the null character.) The generic pointer that `malloc` returns will be converted to `char *` when the assignment is performed; no cast is necessary. (In general, we can assign a `void *` value to a variable of any pointer type and vice versa.) Nevertheless, some programmers prefer to cast `malloc`'s return value:

Q&A

```
p = (char *) malloc(n + 1);
```

When using `malloc` to allocate space for a string, don't forget to include room for the null character.

Memory allocated using `malloc` isn't cleared or initialized in any way, so `p` will point to an uninitialized array of `n + 1` characters: