null-terminated partially filled char arrays

Characters

C-I 310: Lecture 11

and the C-string library; see DSO sec. (4.5).

C-strings
6. All data in a program's process is stored and computed in (current) What printers print instead a conversion value

give the correspondence

7-bit ASCII character set given in Appendix A of DOS. This ASCII table
5. The most popular characters, about which most the World agrees, are the

YOUR C/++ implementation.
different 8-bit chars convert to 0 to 255, or to −128 to 127, depending on
4. chars are automatically converted to and from ints. The 256 = 2
3. A char has at least 8 bits, guaranteed. 8-bit chars are almost universal.

So by definition, sizeof(char) = 1.
2. Sizes of C/++ variables are expressed as multiples of the size of a char.
1. A variable of type char can hold a character of the implementation

Programming Language By

2

University at Albany Computer Science Dept.

Characters, and type char

My Reference: The C++ Programming Language
E is 69
....

```c
if (ch == 'E')
    E = 69;
```

... use namespace std;
// So cout means std::cout
#include <iostream>

### System

#### Base 2

Why 69?

Why Letter E?

ASCII Printer

make it add

8 bits

These same

computers in terms of binary digits or bits.

University at Albany Computer Science Dept.

Social convention of the printer manufacturers!
\{ 
  TRUEl
  
  \{ 
    if
    
    \{ 
      N++;
    \}
    
    TRUEl
  \}
\)

(,0) = i

\texttt{white = Quantity white} (p[N]) // Loop invariant: N == \# non-null characters counted so far

size_t N = 0;

size_t strlen(const char *p)

\textit{C-strings} are often accessed through char * type vars.

\texttt{C-strings} are often accessed through char * type vars.

Unix and other system interfaces libaries use C-strings.

\texttt{#include <cstring> Library has very useful functions.}

\texttt{#include <iostream> Facilitates "know about" C-strings.}

\texttt{A C-string is a null-terminated array of char.}

4 University at Albany Computer Science Dept.
{ return target; } 
\*The last copied char was , 0, 
{ 
"\*True\* Again\* : "I==chars copied so far. 
" \*I++; 
\*Whoops, invariant is FALSE now!!! \* 
while 0 != (target[I]=[src[I]) )  \* // tricky: copy first, then test  
\*\*Invariant: "I==chars copied so far. 
  size-1=0; 
\* 
char * strcpy(char target [], const char src [])
It's important to understand "while" precisely...

```c
return target;

while((target[I]=src[I])!=0)
{
    I++;
}
```

NO!

YES!

The control-expression of the while-statement is always done at least once.

The body of the while-statement is always operation copy character.

This is copy (char target[], const char src[])

University at Albany Computer Science Dept.


Coding `pch[1] = 0.;` is *VERY BAD.*

Coding `acast[1] = 0.;` is *OK and might be useful.*

The address of the given constant C-string.

VERY DIFFERENT: `pch` is a *POINTER VARIABLE* and its initial value is `NULL`!

`acast` means something a little different: `acast` is a *CHARACTER pointer*!

BEWARE: In C/C++, char *pc = "INITIALIZE CONTENTS";

Length of the array.

Compiler will count the number of elements in an initializer to fix the length.

* or by code like `char acast[] = "INITIALIZE CONTENTS"` for which the explicitly (as in `char mystring[13];`)

In C/C++, when you declare an array, you must fix the length, either
How much is sizeof(char* pCH), given the declaration char* pCH?

expression in the ( ),

what's sizeof()? It's NOT a function!!! It's a builtin C/C++ operator that
provides, at COMPILE TIME, the size of a TYPE, or of the type of the
variable, for C/C++. OK for C/C++, but noting OK for C++.

32 bit addresses (OK for C/C++, but noting OK for C++)

2 bytes long so they can hold

29

I'm a string of 29 characters

char* pCH = "I'm a string of 29 characters";

using namespace std;

#include <string>

University at Albany Computer Science Dept.
it points to:

But only if you TRUST the values of pch and the array

strncpy(pch, "Wor...
29 ordinary characters plus the 1 null terminating character!

// inside a block...

ACstr is an automatic variable.

assert(sizeof(ACstr)==30)

char ACstr[]="I'm a string of 29 characters."

inside a block... ACstr is an automatic variable
pACstr is a pointer variable initially pointing to a "constant" array. Assert sizeof(pACstr)==4 on many systems, NOT 30.

On the prev. slide ACstr[] WAS DIFFERENT

char * pACstr = "I'm a string of 29 characters";

inside a block. pACstr is an automatic variable
Backward Compatibility:

char [ ] ? and convert it to const char* making the VERY BAD lines illegal!!

Why don't standard C and C++ make the type of "a String literal" be const

others will crash.

or not on different systems.

Hello string might be copied!

strcpy(\"Hello\", \"Bye-Bye\")

strcpy(\"Bye-Bye\", \"Hello\")

strcpy(\"Hello\", \"Hello\")

// or okay bad too.

char \[\] ENOUGH-SPACE

and null-terminated.

compiler-generated static "pseudo-constant" array filled with the given chars.

a string literal IS CONVERTED TO THE ADDRESS OF

or sizeof(A) (or sizeof(A)\n
Except when used like char *A = \"a String literal\";

(annoying C/C++ thing.)

Typing and sending on the Internet a clever string excessively
early, and professor/textbooks teach it. Nothing prevents a client user from
comprehending it's (even though it's
It's a cruel world: NEVER USE GETS (mychararray)!

So their design choices were rational.

But they did think that every last microsecond of computer time was valuable.

Some nasty people will make that happen on purpose.

The earliest C/UNIX/Internet/DOS/windows designers did not imagine that

OVERFLOW!

When you copy a C-string without counting the characters so that cannot

data.

The other is inadvertent executing of

SOFTWARE EXPLOITATIONS The other is inadvertent executing of

enables people to write VIRUS, WORMS, and other

C-strings are A LEADING VULNERABILITY.
Both: NO MORE THAN BSIZE-1 chars are read; a \0 is copied after the
read chars.

*/ Like cin\gets tells() */
\gets \gets (BUFR, BSIZE, stdin) \gets NEVER \gets (BUFR) \gets EVERY
...\gets
#include <stdio.h>
C-library...
[
These will NOT overflow BUFR]
\gets \gets (BUFR, BSIZE); \gets unformatted, stops after \n
\gets \gets (BUFR); \gets reads a token, skips whitespace
\gets \gets (BUFR); \gets
...\gets
char BUFR[BSIZE];
const unsigned int BSIZE = 0xB1AB;

What you should use:

University at Albany Computer Science Dept.
INTERNET WORM DID THIS on purpose!

If the target array is automatic, these locations are lower in the activation record stack, and so function return info might be corrupted. THE FIRST

those illegally occupied by the target array

MEMORY located at addresses larger than (i.e., after)

chars at or after src's strcopy OVERWRITES THE

When the target array is SMALLER than $1 + \#\text{null}$

DANGER!!
from outside your program!

don't trust psource[i] if it comes
destination array. if

source C-string is > the number of bytes in the

SOMEHOW, you better make sure the length+1 of the

strcpy(pdest, psource);

located by pdest

the C-string located by psource to the array

the following string library function COPIES

... somehow, pdest and psource are assigned values.

... // char *pdest; char *psource;

using namespace std;

#include <cstring>

another survival tip for a cruel world:

university at albany computer science dept.
The `strcpy()` function is similar, except that not more than n bytes of `src` are copied. Thus, if there is no null byte among the first n bytes of `src`, the result will not be null-terminated.

The copy destination string `dest` must be large enough to receive the string may not overlap, and the pointer to `by` `dest`. The strings pointed to are not overlapped, and the array (including the terminating `\0`, character) to the array.

```
#include <string.h>

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

`strcpy()` is a better choice than `strncpy()`.

C-strings.

So, well-written software tracks the length of every destination array for null-termination.
SYID 3, POSIX, BSD 4.3, ISO 9899

CONFORMING TO

Favorite cracker technique.

match happens. Overflowing fixed length strings is a
thing anything failed to check the size before copying
then enough (that is, if the programmer was stupid/lazy, and
If the destination string of a strcpy() is not large

BUGS

, the remainder of dest will be padded with nulls.

In the case where the length of src is less than that of