Characters, C-strings and a cstring

Singly linked lists: DSO Ch. 5

Project 2 Data Str. Diagram

IISUNIX:~acs1310/proj2/command-study.cxx

Command LINE Processing: See

CSI 310: Lecture 9
MySequence pointers, counters, temporary HEAD TAIL, etc. what you design.

New York City, USA

Cape Town, South Africa

Timbuktu, Mali

Name Lat Long

- 73.92 18.37
- 3.07 16.75

Project 2 Core Data Structure Diagram
3. struct node for the **linked list** of landmark data.

2. class landmark for (name, longitude, latitude) data.

1. C-strings for names.

**Project 2 Dynamic objects:**

The dynamic memory used shrink as well as grow as needed, dynamically.

partially filled array.

dynamically allocated. DSO Ch. 4 covers the **dynamically allocated** data structures can be virtually unlimited in size if the objects are objects.

A linked data structure consists of some structure type that contain some pointer type fields that hold addresses of structure type objects.
multitudes of Java Library and Language Classes are dynamic.

ETY: ALL of the objects (variables, instances) of the interesting and full featured

(malloc and free in C).

coded operation (new and delete) executed when the program runs.

But, a dynamic variable is created or destroyed only by an explicitly

function is called and are destroyed (storage recycled) when it returns.

3. CS120I Local extract (C/C++ automatic) variables are created when a

input data can determine if and how many dynamic variables are created.

2. Dynamic variables are created during process execution. Unpredictable

But they, like all variables, have memory addresses.

1. Dynamic variables are not declared. They are not named by identities.

(You need pointers to access them! In Java you need „references“.

Dynamically Allocated Variables

University at Albany Computer Science Dept.
To allocate an array of say 4 ints, do the same except use the array form for

\[ p1 = \text{new int}[4]; \]

new

To allocate an array of say 4 ints.

\[ p1 = \text{new int}; \]

And then make program execute:

\[ \&\&.\text{int} \ast p1; \]

\&e. int *p1;                           

variable to hold its address,'

To allocate ONE variable of any type, say int, make sure you have a pointer

Do int dynamic allocation: use new.
class node { public:
  node * next ptr;
  node * head ptr;
  ...;
};

class node * link ptr;
node * head ptr;

private:
  value type data field;

... and access methods

typedef double value type;

public:
  ... constructor, manipulator

KNOWN LOCATIONS
It consists of nodes, a head pointer, and (usually) a next pointer.

If this set of nodes, a head pointer, and (usually) a next pointer
are very fast (in fact, (small) constant time).

KNOWN LOCATIONS, so entity insertion and deletion near entries at
one-dimensional sequences, so entity insertion and deletion near entries at

Linked List: Concrete Linked data structure Good for Implementing
Building a 1-Item Linked List

Classic C-style

```
struct node
{
    double data;
};
```

OOP/Modern C++ Style

```
class node {
    public:
        node(const val-type v = val-type(),
            typered double val-type;

    struct node * head;
    struct node * tail;
```
With our classical style code, (7) and (9) are executed before (1). (€)

```
(€) Copy "ditto" into automatic variable HEAD.
(9) Copy the address (pointer to the new node) into automatic variable TAIL.
(7) Copy 0 (NULL) into the link field of it.
(9) Copy (short) 23.6 into the data field of the new node variable.
(7) Run the dynamic memory allocator "new" of the C++ standard library.
```

For both styles, the computer does the same things with the same results.
// Copying takes place before you lose your
HEAD = tp;

HEAD = HEAD;

if (HEAD==NULL) //

HEAD = new item;

struct node tp;

double data;

class node

OO|/Modern C++ Style

Adding an item to the front:
The old value of HEAD was the ADDRESS of the node „housing“.

(1) Run the dynamic memory allocator „new“ of the C++ support library.

(2) Copy (malloc) 14.7 into the data held of the new node variable.

(3) Copy pointer value from HEAD into the link held of the new node. (This

variable HEAD gives TAIL its value is the address of the node containing 23.6.)

(4) Copy the address of the new node (the one containing 14.7) into automatic

variable in HEAD, (This operation OVERWRITES the previous value in HEAD, so it

must be done AFTER (3).)

(5) If TAIL==NULL, the original list was empty, so TAIL=HEAD gives TAIL its

 correct value!

23.6; it was NOT 23.6.
meaning, plus practice solving problems! understanding of variables, data and (C/C++) language

Have patience: Speed and elegance come from precise

these steps.

If it fails for boundary cases, try to fix it and repeat

(m and m-1em list)

Then, check it for any boundary case(s). (E.g. Empty

and draft code if not.

Check that it works in the general case. Re-do design

idea in the general case.

Advice: Try to design and draft code for an algorithm
otherwise. It's good!

This last C++ statement works when the list was empty, and does nothing.


```c

if (iHEAD) (HEAD = TAIL;)
    TAIL = t;

if (TAIL) TAIL->LINK = t;

...;

if (TAIL) TAIL->LINK = NULL;
t->LINK = NULL;
t->DATA = 33.3;
node * t = new node;

...

void set-TR(node * p);
    node * t = p;
    t->LINK = A;

node (const val-type v = val-type);
typed double val-type;

typedef double val-type;

class node

Classical C Style

OO/Modern C++ Style

Adding an item to the end: Requires modifying last node.
```
TAIL before dereferencing it, to set the last

// woe is you if you lose your
$ \text{if (ptr != NULL)}$

... can be shortened to $\implies \text{if (ptr)}$.

I. Since the last node of the original list must be modified (to replace the

NULL link with the address of the new node), the code for modifying

MUST be conditional on whether there is a last node.

$\text{when isn't there a last node?}$

2. Rule of $C++$: The NULL pointer value is converted to boolean False

inside conditionals; all non-NUL values convert to True there. So, for

etc. etc.
node *t = new node(33.3);
if (!HEAD) HEAD = TAIL;
if (TAIL) TAIL->set_lk(t);
TAIL = t;

if (TAIL) TAIL = t;
if (TAIL) TAIL->set_lk(t);

node *x = new node(33.3);
TAIL = t;
{
    HEAD = t;
    /* list is empty */ assert(HEAD);
}
else
{
    TAIL->TINK = t;
    /* list is not empty */ assert(HEAD);
}
if (TAIL)
    list is empty:
Use one conditional to control all operations depending on whether the original

{ if (HEAD) HEAD = TAIL;
   TAIL = t;
if (TAIL) TAIL->TINK = t;
if (TAIL) TAIL = t;
   t->TINK = NULL;
   t->data = 33.3;
node * t = new node;

Algorithm Improvements from:
University at Albany Computer Science Dept.
and the C-string library, see DSM Sec. 4.5.

C-strings

Null-terminated partially filled char arrays

Characters
what printers print \rightarrow \text{int conversion value}

give the correspondence

7-bit ASCII character set given in Appendix A of DOS. This ASCII table

The most popular characters, about which most the World agrees, are the

YOUR C/C++ implementation.

different 8-bit chars convert to 0 to 255, or to −128 to 127, depending on

chars are automatically converted to and from ints. The 256 = 2^8

A. A character has at least 8 bits; guaranteed, 8-bit chars are almost universal.

So, by definition, sizeof (char) = 1.

2. Sizes of C/C++ variables are expressed as multiples of the size of a char:

A \text{ character is a character literal.}

I. A variable of type char can hold a character of the implementation

Programming Language

The C++ Programming Language

by

University at Albany Computer Science Dept.
Computers in terms of binary digits or bits.

These same 8 bits sent to an arithmetic adding unit make it add

<table>
<thead>
<tr>
<th>01000101</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 32 16 8 4 2 1</td>
</tr>
</tbody>
</table>

Why letter E? Social convention of the printer manufacturers!

Why 69? Base 2 numeral system: 1+4+64

```
#include <iostream>  // Get definition of std::cout
using namespace std;  // So cout means std::cout
...
char ch = 'E';
int I = ch;  // char to int conversion.
cout << ch << " is " << I << end;
...
E is 69
```
```c
{
    /*
      * TRUEN || return N;
      */
    {
        i++;
        if (N++;
            if (N++)
            if (i++)
                if (0) i = 0;
            } // C-string are often accessed throughout char * type vars.

        // Unix and other system interface libraries use C-strings.
        #include <cstring>
        
        // Library has very useful functions.
        #include <iostream>
        
        // A C-string is a null-terminated array of char.
    }

    // ++++ TRUEN || non-NULL chars COUNTED so far
    // LoopInvariant: N==#non-null chars counted.
    // size_t strlen(const char *p)
    
    // C-strings are often accessed throughout char * type vars.
    
    // Unix and other system interface libraries use C-strings.
    #include <cstring>
    
    // Library has very useful functions.
    #include <iostream>
    
    // A C-string is a null-terminated array of char.
    /*
      */
    }
```
{ return target;

  // The last copied char was '0',
  }
  return(TRUE);
  }
  // (true) again.
  ++I;

  // Whoops, it is FALSE now!
  while (0 == (target[I] == src[I]))
    // tricky: copy first, then test
    size_t I=0;

  } char * strcpy(const char target[], const char src[])

It's important to understand "while" precisely...

```
return target;
```

```
target[I] == 0?
I++;
```

```
target[I] = src[I];
```

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

```
strcpy(char target[], const char src[])
```

```
I = 0;
```

```
((target[I] = src[I]) == 0)
```

```
NO!
YES!
```

```
The control-expression of the while-statement
The body of the while-statement
```

```
The body of the while-statement
```

```
This character operation is always done at least ONCE!
```

```
This copy of the character
```

```
The while-statement of the while-statement
```

```
University at Albany Computer Science Dept.
```
coding pch[1] = 0; is VERY BAD.
coding acs[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
BEWARE: In C/C++, char *pch = "InitialContents"; means something

length of the array.

compiler will count the number of elements in an initializer to fix the
or by code like char acs[] = "InitialContents" (for which the
explicitly (as in char *string[13];)

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

expression in the (.),

what's `sizeof()`? It's NOT a function!!! It's a built-in C/C++ operator that
provides, at COMPILE TIME, the size of a TYPE or of the type of the
variable.

32 bit addresses (OK for 4 GiB开辟); 32 bit virtual memory's (Why?)

`sizeof(char) == 29` 4 is NOT `strlen(pch)`
I'm a string of 29 characters

```
str = strlen(pch) == 0 >>

"" = "" >>

"" = "" >>
```

...
strzecf (pch)

Words for the wise: Always use strlen(pch)+1 to get how many bytes a C-string occupies; never use
Looking back on the Internet, clever string execution
Typing and sending on the Internet a clever string excessively
easy and professors/textbooks teach it. Nothing prevents a client user from
It's a cruel world: NEVER USE GETS (mychararray); EVEN
(j)
So, their design choices were rational.
But they did think that every last microsecond of computer time was valuable.

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

OVERFLOW!
A string does not exceed the size of the destination, there will be a so-called BUFFER
When you copy a C-string without counting the characters so that count

SOFTWARE EXPLOSIONS!
Enables people to write VIRII, WORMS, and other G-strings are a leading vulnerability that

University at Albany Computer Science Dept.

28
read chars;

Both: no more than \n \n \n \n \n \n

/* LIKE cn\n\n*  */

gets(buffer, size); /* NEVER gets(stdin); EVER!! */

... include <stdio.h>

C-library....

[]

These will not overflow buffer

\n \n cn\ngets(buffer, size); /* unformatted, stops after \n cn \n \n \n cn \n \n \n \n \n \n \n \n \n \n \n \n \n
What you should use:

const unsigned int size = 0x1ab;

University at Albany Computer Science Dept.
INTERNET WORM did this on purpose!

If the target array is automactic, these locations are lower in the activation record stack, and so function return into might be corrupted. The first those illegally occupied by the target array.

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

when the target array is Smaller than 1+non-null

DANGER!!
from outside your program!

don't trust your destination array. If it comes

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

strcpy(dest, source);

located by dest

the C-string located by source to the array

The following string library function COPIES

... somehow, dest and source are assigned values.

... //

cchar *dest; cchar *source;

using namespace std;

#include <cstring>

Another survival tip for a cruel world:
29 ordinary characters plus the 1 null terminating character!!

assert(sizeof(ACstr)==30)

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

Inside a block... ACstr is an automatic Variable.
assert(sizeof(pACstr)==4) on many systems. NOT 30.

pACstr is an automatic variable.

On the prev. slide: ACstr WAS DIFFERENT
char * pACstr = "I'm a string of 29 characters";

Inside a block: pACstr is an automatic variable.

pACstr is a pointer variable initially pointing to a "constant" array.
"a''r''a''c''t''e''r''s''\0"
Backward Compatibility?

Here is an example: making the VERY BAD lines illegal.

```c
char [ ]

and convert it to const char * making the type of "A String Literal" be const

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 Greeting might be copied!

strcpy("Greeting", "Bye-Bye");

strcpy("Greeting", "Hello"); // VERY BAD too.

strcpy("Greeting", "Hello"); // Very BAD

strcpy("Greeting", "Hello"); // OK but Dangerous.

...

char Greeting[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 String Literal") (or sizeof(AA))

or sizeof("A String Literal")

Except when used like char AA [] = "A String Literal";

Annoying C/C++ thing:

(Annointing C/C++ thing:

University at Albany Computer Science Dept.
new is a keyword in C++. See DSO FIGURE 4.1

    PAT = new throttle[5];
    throttle *PAT;

To allocate an array of 5 throttles:
Revised Bag Class

```java
class Bag {
    private int capacity; // Current capacity of the Bag.
    private int used; // How much of the array is used.
    private Object[] data; // Pointer to dynamic array.
}
```
Pbag is a pointer to a structure. Structures can contain pointers to names. Dynamic vars. have no

Unlike the auto vars, variables,

Pbag = new bag2(4);

Pbag

AutoBag

AutoBag. data is a pointer to an array.

#include "bag2.h" //Ch.4 improved bag

main()

#include "bag2.h" //Ch.4 improved bag