The dynamic memory used shrinks as well as grows as needed, dynamically allocated.

Such data structures can be virtually unlimited in size if the objects are

A linked data structure consists of some structure type objects (variables) that

linked lists: DOS 3.5

Lab 3 and Prof available now.

CSI 310: Lecture 8
pointer to (addr of) a char

pointer to (addr of) a node

NULL

NULL

NULL

This line's been cut

Another cut

First main list text line

2nd line

The third

Project 2 Core Data Structure Diagram

myeditorCore

cursor and any others you design

cut HEAD

cut TAIL

NULL

pointer to (addr of) a char
node * tail-ptr;
node * head-ptr;

...;

private:
value-type data-field;

... and access methods

constructor, manipulator

typed double value-type;

public:

class node { public:
    double data-field;
    node * link-field;

    // equity: strict node
    node (node * pnode = NULL);

    // equality:
    node & operator=(node & pnode);

    friend class node;

    ~node();

private:

class node { public:

    node (node * pnode = NULL);

    // equality:
    node & operator=(node & pnode);

    friend class node;

    ~node();

private:

    node * tail-ptr;
    node * head-ptr;

    ...;

    node * link-field;

    value-type data-field;

    ... and access methods

    constructor, manipulator

    typed double value-type;

    public:

    class node { public:
        private:
```c
// these 2 lines
node = new node(23, 6);

node * HEAD;
node * TAIL;
node * TAIL;
node * HEAD;

struct node

Classic C Style

class node

OOP/Modern C++ Style

Building a 1-Item Linked List
```
With our classical style code, (7) and (5) are executed before and (9) after.

1. Copy "ditto" into automatic variable HEAD.
2. Copy the address (pointer to the new node) into automatic variable TAIL.
3. Copy 0 (NULL) into the TAIL field of it.
4. Copy (3) into the TAIL field of the new node variable.
5. Copy (23.6) into the data field of the new node variable.
6. Run the dynamic memory allocator "new" of the C++ Support Library.
7. For both styles, the computer does the same things with the same results.
```
if (TAIL == NULL) TAIL = HEAD;

// HEAD before copying it!
if (TAIL == NULL) HEAD = tp;
    HEAD = tp;
    tp->link = HEAD;
    tp->data = 14.7;
    tp = new node;
    node * tp;

struct node

```

```
if (TAIL == NULL) TAIL = HEAD;

HEAD = new node (14.7, HEAD);

private:

node * TINK;
val-tp data;
private:

...


data = p;

{ node = p * NULL;
node (const val-tp & val-tp)
}

typedet double val-tp;

public:

class node

```

```
00f/Modern C++ Style

Adding an item to the front:

```
23.6; it was NOT 23.6.

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 (statt) 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 is the previous value in HEAD, so it must be done AFTER (3).
(4) Copy the address of the new node (the one containing 14.7) into automatic value is the address of the node containing 23.6.
(5) If TAIL==NULL, the original list was empty, so TAIL=HEAD gives TAIL its (old value before step 4)
correct value.

The old value of TAIL was the node "data link" 23.6.

(1) Run the dynamic memory allocator "new” of the C++ support library.
(2) Copy (statt) 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 is the previous value in HEAD, so it must be done AFTER (3).
(4) Copy the address of the new node (the one containing 14.7) into automatic value is the address of the node containing 23.6.
(5) If TAIL==NULL, the original list was empty, so TAIL=HEAD gives TAIL its

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

Have patience: Speed and elegance come from precise
these steps.
If it fails for boundary cases, try to fix it and repeat

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 (HEAD) } HEAD=TAIL;
TAIL=t;
if (TAIL) TAIL->TAIL=t;

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

    { void set-Tk(node * p) { tink=p;
        node * tink;
        double data;
    }
    } } struct node

classical C Style

```
TAIL before dereferencing it, to set the last think!
... (pt;=NULL)

... can be shortened to (pt) if

inside conditionals. All non-NULL values convert to True there. So,

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

When isn't there a last node?

Must be conditional on whether there is a last node

NUL link with the address of the new node, the code for modifying it

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

Remarks:
node *t = new node(33.3);
if (!HEAD) HEAD = TAIL;
if (HEAD == TAIL) does nothing here.

if (HEAD != TAIL)
    node *t = new node(33.3);
    t->link = TAIL;
    TAIL = t;
    if (HEAD == TAIL)
        TAIL = t;
    TAIL->set_lk(t);
TAIL = t;
{
  HEAD = t;
  if (list is empty)"assert (HEAD);
  else
  {
    TAIL->next = t;
    if (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) TAIL=HEAD;
      else TAIL = t;
      if (TAIL) TAIL->next = t;
      t->next = NULL;
      t->data = 33.3;
      node * t = new node;
      if (HEAD)
Intro. C-Strings Sec. 4.5

Pointers to structures.

)))) (Copying structs. with pointers: Characters, char.

Characters, char and binary.

Pointer Terminology.

CSI 310: Showed out Lecture 8
Or is it the value of a pointer variable, that is, an address?

So, when you or others say ‘pointer’, think hard: Is it a pointer variable?

But most everyone, we and DSO, say, for short, ‘PIVAR is a pointer’

(illegal value)

C/C++ int variable. Or else it might have the NULL value, or else some
pointer type variable. The variable named PIVAR might store an address of a
we and DSO said "a pointer is an address". PIVAR is really (the name of)

int *PIVAR; What is PIVAR? Is it a "pointer"?

int PIVAR; What is int? What is int PIVAR?

int int; What is int PIVAR?

A linguistic pitfall—try not to fall into it!
Perhaps we should always use the word "address" for "pointer value".

Done.

Type which determines what values it can hold and what operations can be performed. Each variable has a type. Technically, "pointer" and "int" describe C++ types. Each variable has a
char type

`null-terminated strings` or `C-strings`
6. All data in a program's process is stored and computed in (current)

what printers print | int 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/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 = 2^8

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

\forall \text{ char } c, \forall \text{ character set } S, c, \text{ and type char }

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

Strongly typed C++ is creator.

Characters', and type char

Programming Language

University at Albany Computer Science Dept.
Why letter E?

Base 2

Why 69?

69

Base 2

make it add

unit unit

adding arithmetic arithmetic

8 bits

8 bits

These same

These same

ASCII printer

make it print sent to an

computers in terms of binary digits or bits.

Social convention of the printer manufacturers!

Why letter E?

Base 2

numeral

system

1+4+64

E is 69

...
{  return N;  
    }  

    }  

    }  

    }  

    }  

    }  

    }  

    }  

    }

#include <cstring> // Library has very useful functions.

#include <iostream> // Facilitates "know about" C-strings.

A C-string is a null-terminated array of char.
{ return target;
  if (last == 0) // The last copied char was , 0
  { true(again).
    i++;
  } // Whoops, it is FALSE now!!
  i = (target[I] == src[I]) // tricky: copy first, then test
    while (i && #chars copied so far.
    size-t I=0;
  }
}

char * strcpy(char target[], const char src[]){
It's important to understand "while" precisely...

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

This is always done at least ONCE!

I++;

The body
```

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

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

```
YES

target[I][I]=src[I]?

COPY character
```

```
YES

target[I][I]=src[I]?

while(0!=(target[I][I]=src[I]))

I = 0;
```

```
NO

target[I][I]=src[I]?

This is always operation copy
```

```
NO

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

```

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

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

If the target array is autonomic, those locations are lower in the activation 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)

characters or after sc, strcpy OVERWRITES THE

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

DANGEROUS
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
On the prev. slide, 

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

was different

inside a block, pACstr is an automatic variable

pac
cstr is a pointer variable initially pointing to a "constant" array

assert(sizeof(pACstr)==4) on many systems. NOT 30.
Backward Compatibility!

Why don't standard C and C++ make the type of "a String literal" be const... other wise will crash.

or not... on different systems...

Hello "greet" might be copied!

... char greet[ENOUGH-SPACE] = 'not-terminated. 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 (AA))

Except when used like char A = "a String literal";

(annoying C/C++ thing...)