array

array

array

array

array

array

array

array

array

array

array

array

array
A Dynamic Variable is useless without a (separate) pointer variable to hold the address returned by new!

<table>
<thead>
<tr>
<th>Dynamic struct</th>
<th>Automatic struct</th>
<th>Structure/Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ new arr[5z][];</code></td>
<td><code>{ baf2 MPG2(zz);</code></td>
<td>class</td>
</tr>
<tr>
<td><code>fun(sizet sz)</code></td>
<td><code>fun(sz);</code></td>
<td></td>
</tr>
<tr>
<td><code>include &quot;bagz.h&quot;</code></td>
<td><code>include &quot;bagz.h&quot;</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic array</th>
<th>Automatic array</th>
<th>Array</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ ... new int[z];</code></td>
<td><code>{ ... int x[96];</code></td>
<td>Arrary</td>
</tr>
<tr>
<td><code>fun(sz);</code></td>
<td><code>fun();</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic integer</th>
<th>Automatic integer</th>
<th>Scalar (Integer)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ ... new int;</code></td>
<td><code>{ ... int x;</code></td>
<td></td>
</tr>
<tr>
<td><code>fun();</code></td>
<td><code>fun();</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic variable</th>
<th>Automatic variable</th>
<th>More Orthogonality</th>
</tr>
</thead>
</table>
structures: Remember Table 2! Structures beyond Ch. 3? Tricky?

PBag is a pointer to a structure. (Structures can contain pointers to
AutoBag's data. data is a pointer to an array.

names.

Unlike the auto. variables, dynamic vars. have no
unused

capacity
used
data

capacity
used
data

AutoBag

belonging to this call to main()

Automatic variables

belonging to this call to main()

heap' or 'free store' of dynamic variables
new bag2(4);

PBag = new bag2(4);

bag2 *PBag;

PBag = new bag2(4);

bag2 AutoBag(4);

bag2 AutoBag

AutoBag

main()

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

It gets complicated!
University at Albany Computer Science Dept.
2. Illegal Pointers:

RECYCLED memory!

1. DANGLING Pointers: Pointers to Garbage or

2. (2) Big Pitfalls of Pointers:

will create a DANGLING POINTER (value in page)

delete page;

Now,
```
main()
#include "bag2.h" //Ch.4 improved bag

bag2 *PBag;      //Pointer variable!
PBag = new bag2(4);

delete PBag;
```

This is a "heap" or "free store" of dynamic variables belonging to this call to main()

Automatic variables

```
 bag
 | 4 |
 | 0 |

PBag
```

```
AutoBag
 | 4 |
 | 0 |

PBag
```

```
main()
#include "bag2.h" //Ch.4 improved bag

PBag = new bag2(4);

```

```
d = new variable (4)
```

```
AutoBag
 | 4 |
 | 0 |

PBag
```

```
AutoBag
 | 4 |
 | 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```

```
AutoBag
| 4 |
| 0 |
```
This is a "CRASH": Computer tried to read memory at the illegal address 0x0,

so it could try to print the int value located there.

Segmentation Fault

```
cout >> *MyP;  
0x0 0xb004e044

cout >> "  "  
cout >> MyInt;  
cout >> "  "  
cout >> MyP;  
4006

cout >> MyInt;  
```

After Assignment:

Before Assignment:

The NULL pointer value is very useful to copy and test, but is illegal to dereference.

University at Albany Computer Science Dept.
Remember: The assignment operator "=".

```
#include "base2.h"

main() {
    main
    }  

baze2 MYbase2;  
MYbase2 insert = (9);  
MYbase2 insert = (9);  
baze2 othbase2(4);  
baze2 othbase2(4);  
} 
```
Something bad happens!

MyBag1.erase(−9);
OthBag1.erase(−9);
OthBag2.erase(−9);
OthBag1.insert(7);
OthBag2.insert(7);

OK, each bag was copied. Suppose these operations next:

Ok, each bag was copied. OK
more important for CS1310. C-standard, linked structures, efficiency and recursion. Study 4.3-4.4 for details. Advanced C++ features help you create pointers!

MORAL: Beware of copying data structures with: 

```c
...

base(const base src);

void operator=(const base src);

public:

class base

 assignment operator and copy constructor

 How? Declare and implement your class's own

 base2 T(base2(base2 MBa2));

otherbase2 = MBa2;

code you write runs for copy operations like: 

Unlike C', C++ gives you programmers the capability to program a class so

University at Albany Computer Science Dept.
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".

(If illegal value.

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 variable named "PIVAR is an address". PIVAR is really (the name of) a variable whose type is "int". This variable stores a C/++ integer.

Most say "It's an integer!" but, really, IVAR is (the name of) an variable.

int IVAR; What is IVAR?

A linguistic pitfall—try not to fall into it.

University at Albany Computer Science Dept.
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

Technically, "pointer" and "int" describe C/C++ types. Each variable has a
These words, the allocated space MUST NOT BE

EXCEED!!

For both ways, the allocated space MUST NOT BE

183-186

Example: C-string pages

data structure = array (variable) + "used" count variable.

1. Data structure = array (variable) + "used" count variable.

2. Data structure = the array ONLY. The end of used prefix is marked by the

next entity containing a terminating value. Example: C-string pages

?? Where does the used data prefix end? Two ways to tell:

array is "idle".

The used data is stored in a consecutive prefix of the array. (The rest of the

Partially filled array: dynamically allocated or not.
Real decision-makers KNOW and COMPARE advantages and disadvantages. Science seeks to discover what is possible, so

Item values are not restricted; No special terminating value is required.

Append a new item to the end; is Fast (also constant time).

You can tell the number of items right away, "in constant time."

A single pointer locates the array:

Beginning to end.

It's the fastest way to control sequential processing of the array from

Advantages of Way 2: If you can live without the terminating value:

University at Albany Computer Science Dept.
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 DSO. This ASCII table
5. The most popular characters, about which we must 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:

\[
\forall \text{ char } ch = \forall \text{ character literal.}
\]

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

#include <stdio.h>
#include <string.h>

int main()
{ }

Characters, and type char
My reference: The C++ Programming Language by
University at Albany Computer Science Dept.
Why letter E?

69

Base 2

Why 69?

System numeral

0
I
0
0
0
0
0
0
*1
*2
*4
*8
*16
*32
*64
*128
*256
*512
*1024
*2048
*4096
*8192
*16384
*32768
*65536
*131072
*262144
*524288
*1048576
*2097152
*4194304
*8388608
*16777216
*33554432
*67108864
*134217728
*268435456
*536870912
*1073741824
*2147483648
*4294967296
*8589934592
*17179869184
*34359738368
*68719476736
*137438953472
*274877906944
*549755813888
*1099511627776
*2199023255552
*4398046511104
*8796093022208
*17592186044416
*35184372088832
*70368744177664
*140737488355328
*281474976710656
*562949953421312
*1125899906842624
*2251799813685248
*4503599627370496
*9007199254740992
*18014398509481984
*36028797018963968
*72057594037927936
*144115188075855872
*288230376151711744
*576460752303423488
*1152921504606846976
*2305843009213693952
*4611686018427387904
*9223372036854775808

This is 69...

E is 69...

count << ch << " " << I << " " end;

int I = ch; // char to int conversion.

char ch = 'E';

using namespace std;

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <iostream>

#include <i...
0->T
...  

{count ++ ch ++ "" I->T
  \n  I = ch;
  char ch = \0;  
}

#include <iostream>

using namespace std;

as the null character for assignment to char variables.

The C/C++ char literal \0 signifies the constant 0. So, \0, 0, and

The null character

from integer value 0.

Has bit representation 00000000, so the null character converts to and

University at Albany Computer Science Dept.
You must learn about them and will use them in Project 2.

Linux and other system interface libraries use C-strings.

```c
#include <cstring>
```

Furthermore, the `strlen` library has very useful functions.

```c
#include <cassert>
```
C-strings are different from C++ strings you get from #include <string>

char [M][0 == [M][4] = M) => [M][3] = [M][2] = [M][1] = [M][0] = M; 

The C-string "ABCD" (4 letters) is stored in a LENGTH 5 (give, not 4)
string in char array terminated with \0, are called C-strings

The null char is coded \0, "A" is coded \'<

In C/C++ the char \'< the char \"
Unix and other system interface libraries use C-strings:

```c
#include <cstring>
#include <iostream>
#include <fstream>
```

The C-string implementation consists of the array and its address.

An `array` of non-null characters whose end is delimited by the "null" `char` value.

This very popular "legacy" C-string data structure is (just) a partially filled and

Finally, `d`.

prints the characters in order, `H` then `e` then `l`'s two `l`'s then `o` then `` space `the` then `W` then `o` then `r` 

`count` is the "Hello World".

is easier to think about than

prints the string "Hello World".

The C-string:

end is marked by the "null character" (is a single variable that holds ONE

It is useful to think that an array of `char` variables whose

University at Albany Computer Science Dept.

On
What does the computer know when to stop printing?

```cpp
    count >> MYCHAR; end:
    Printing what you typed:

    It always uses the last for \

    (The 12 tells ostream: GETTIME() to use only up to 12 chars in MYCHAR.)

    cin.gettime(MYCHAR, 12);
    Reading up to 11 characters you type on one input line:

    //Holds a C-string with length up to 11
    char MYCHAR[12];

    REQUIRED in CS1310: Declaring a variable that can hold a C-string:
    count >> "HELLO WORLD";

    C-strings are very easy to use. You have used them in CS121 code like:
    using namespace std;

    #include <iostream>
```
```cpp
{ 
    count >> BST.size reports: "size of ACST: " count >> last char in ACST has int value: " last >> count end; 
    int last = ACST[count]; 
    count >> count end; 
    count >> length of ACST >> ACST >> "=" " = \n    { count+=
        else
    break; // terminate the while loop
    if (ACST[count] == 0) {
        }
    while(true)
    int count = 0;
    char ACST[] = "I'm a string of 29 characters!
    main()
using namespace std;
#include <iostream>

University at Albany Computer Science Dept.
22
BUT size of reports: 30
Last char in Acceptor has int value: 0
Length of "I'm a string of 29 characters" = 29
29 ordinary characters plus the 1 null terminating character:

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

assert(sizeof(ACstr)==30)
coding `pc[i] = 0;` is VERY BAD.
coding `acst[i] = 0;` is OK and might be useful.
the address of the given constant C-string.

VERY DIFFERENT: `pc` is a POINTED VARIABLE and its initial value is

BEWARE: In C/C++, char *pc = "InitializeContents"; means something

Length of the array.

compiler will count the number of elements in an initializer to fix the

or by code like char acst[] = "InitializeContents" 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(char)`? It's NOT a function!!! It's a built-in C/C++ operator that provides, at COMPILATION TIME, the size of a TYPE, or of the type of the

32 bit addresses (OK for a quadbyte sized virtual memories).

Why?? Pointer variables (today) are typically 4 bytes long so they can hold

```cpp
int sizetof(pCH) = 4; is NOT strlen(pCH) = 29
```

I'm a string of 29 characters

```cpp
'strlen(pCH) >> \\
'' = is NOT strlen(pCH) >> \\
'sizetof(pCH) >> '' = sizetof(pCH) >> \\
```

```cpp
cout >> pCH >> std::endl; cout >> pCH >> std::endl; char * pCH = I'm a string of 29 characters''
```

```cpp
using namespace std;
```

```cpp
#include <string>
```
Words for the wise: Always use strlen(pch)+1 to get how many bytes a C-string occupies; never use strlen(pch).
Typing and sending on the Internet a clever string excessively

Nothing prevents a client user from easy and professors/textbooks teach it.

It's a cruel world: NEVER USE GETS (MYCHARTER)? EVER!! (even though it's

So, their design choices were rational. (j)

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 designs did not imagine that

OVERFLOW!

doesn't 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

data.

SOFTWARE EXPLOITIONS! The other is inadvertent executing of

enables people to write VIRII, WORMS and other

c-strings are a LEADING VULNERABILITY that

gee, that's wonderful. Wow!

University at Albany Computer Science Dept.
from outside your program!

don’t trust source [] if it comes
destination array. the number of bytes in the
source C-string is > the length of the

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 functionCopies

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

... //

char *dest; char *source;

using namespace std;

#include <cstring>

Another survival tip for a cruel world:

University at Albany Computer Science Dept.