\[(\text{input}) X)_{H} = 1^{+} X \quad \text{initial conditions } X = 0 \]

\[
\begin{align*}
\text{Discrete Time: } & X(t) \text{ is a sequence of state vectors, } t = 0, 1, 2, \ldots \\
\text{Continuous Time: } & X(t) \text{ is the state vector, a function of time.}
\end{align*}
\]

Physics/Mathematics: Dynamical System

(Propertics)

When a program RUNS, as opposed to when it is compiled or built (static)

Dynamic: Pertaining to behavior that varies in time. Properties determined

CSI 310: Lecture 9
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 contained in some pointer type fields that hold addresses of structure type objects.

A linked data structure consists of some structure type objects (variables) that point to and dynamic memory applied to Partially Filled Arrays: DSO Ch. 4, 5

Linked Lists: DSO Ch. 4, 5

Pointers and dynamic memory applied to Partially Filled Arrays: DSO Ch. 4, 5

Trainz and Pro!z available now.

Pointers/Reference/Object diagrams are drawn on whiteboards in software development offices all over Silicon Valley, NYC, etc.

Dynamically:

Last week’s lab: Dynamic Fill (essence of Chapter 4) — capacity grows
variable in the Node structure.

2. Stored OUTSIDE the Node structure and be referred to via a pointer.

1. Stored RIGHT INSIDE the Node structure.

Application data in each node could be:

Data structure of linked Nodes all the same type.

Data structure holding a C-string pointer.

This overcomes the static capacity limitation of C's and sizeof partially.

Dynamic capacity partially filled arrays.

last used element.

2. C-string strategy: NULL (or other termination value) stored one-past the

1. With "used" variable to record how much is used.

Fixed capacity partially filled arrays.

Fixd length arrays.

(Throttle, Point, Gain).

Simple (scalar) variables: primitive and struct/class

CS1310 Review and new data structures:

University at Albany Computer Science Dept.

© S. Chakravarty 2006
systems belonging to most people.

The huge fixed allocation make the software unusable on the smaller.

The time overhead for allocating huge memory spaces for occasional large

data sets make the software unacceptable slow on most uses.

Because:

(1) Capacity too small: limits size of application data.

(2) Using maximum conceivable capacity is unsatisfactory

Basic Limitations of Static Capacity Data Structures:

© S. Chakravorty 2006

University at Albany Computer Science Dept.
3. Code a constructor and member functions to REPLACE the current array

```cpp
class Pile {
    int number;
    int arrayCapacity;
    Pile* parent;

    public:
        Pile();

    private:
};
```

2. The DYNAMIC Pile has 3 data members:

1. SEPARATE OUT the array of Coims from the Pile structure.

Idea for Dynamic Partially Filled Array:

When it replaces it with a new array,

Such member function MUST copy the used Coims from the old array with a NEW array that is bigger.
Which equals eleven? and 9, and \\032, 0.13

Return value from a function call, and C/C++ literals like 386, 0x3BD, 013

Answers from class discussion: initializer in a declaration, import from the user,

Ask yourselves: How do you program in C++ to "get" an integer value? Some

Alternative array [] syntax:

'Dereferencing pointer variables or values: in an EXPRESSION

- Pointer fundamentals:

  • Obtaining pointer (i.e., address) values: [g, declared array name, new,

  • Declaring pointer variables: in a DECLARATION.

  • Matto() or other function that returns a pointer value.

  • Dereferencing pointer variables or values: * in an EXPRESSION.
Some utterly USELESS code:

...;
    new int;
...

WHY?

The new operator

1. Dynamically allocates one object or variable of the type or class specified.
2. Calls the appropriate C++ constructor if one has been defined for the class.
3. RETURNS the ADDRESS OF (in other words, THE POINTER TO) the object or variable so allocated.
Like this is a CORE LEARNING OBJECTIVE.
and developing program code to do manipulations

Understanding DATA STRUCTURE DIAGRAMS

Not much happens... until user types I 7 <enter

 cin >> *PMyInteger;

PMyInteger

garbage

variable is created
an anonymous integer

PMyInteger = new int;

double \[4\]

garbage

int * PMyInteger;

C.S. Chahkiyen 2006

University at Albany Computer Science Dept.
The Third Line's been cut
Another cut
This line's been cut
First Main List Text Line
2nd line
The Third line

Project 2 Core Data Structure Diagram
changed.

Features of the variable ‘is VALUE’ (synonym: state) can be.

‘clone’ person, automobile, shoebox, a variable has a unique identity even after it is

The NAME can be copied. The VARIABLE IS THE OBJECT ITSELF LIKE a

VARIABLE

a NAME can be the VALUE of a

4. This is new.

conceptual entities (3 different things).

3. The value, the name, and the variable itself are three DIFFERENT

2. Each variable holds a value (or state).

1. Each variable (synonym: object) has (some kind of) name.

The main points...
Brandon = 98.
variables are comprised of different numbers of bytes. Don’t worry about this yet...

One big house might cover several adjacent lots. Technically, different sizes of

current VALUE,

from the variables’ memory locations. A segment of computer memory consists of numbered memory

house’s current OCCUPANTS.

called address. The address of a house's memory locations. A block of real estate consists of numbered houses. The house numbers are

ame that C++ code can copy, use, store, etc., at runtime.

A pointer is the memory address of a variable. A pointer is the kind of
Computer memory is like an array; where addresses function as subscripts. The value of this variable is 57. The variable points to a 4-byte integer variable.

Pointer value or address of the variable 57.
The above integer variable whose address is 992 and value is 57.

The pointer variable whose address is 987 is pointing to

Here's a pointer variable whose value (992) is the address of

Here's the integer variable whose address is 992.

You will see hex in Lab2.

Real programmers write their addresses in hexadecimal.

I will denote pointer values by black dots because the numeric value is usually boring.

Pointer values, like any other numbers, can be stored in hexadecimal (base sixteen) because hex to binary (base 2) conversion is very easy.

Real programmers write their addresses in hexadecimal.

(You will see hex in Lab2.)
POINTER VALUES, and call them "references".

Unlike C/C++, Java, and Perl (except debuggers) hide all numeric

(For more into, "Google IT" and read books like the Hennessy and Patterson

systems. Sometimes within I/O devices (memory mapped ones).

The memory bus addresses usually locate data within cached RAM hardware

addresses and sometimes into page fault signals.

Bus or physical memory management units sometimes into hardware memory

Numerical (binary) virtual addresses are quickly translated by 

Hardware-software interface (CS1333).

Numerical pointer values are virtual addresses, and part of the

PC/Workstation/Servers with Unix-like/Windowns NT operating systems:

a few background words... In late 20th century technology of
The ADDRESS of X is 0xbfffff9b4

The VALUE of X is 98.6

float X = 98.6;
cout << "The ADDRESS of X is " << &X << " The VALUE of X is " << X << endl;

main()
{
#include <iostream>
using namespace std;

To use dynamic data structures in C/C++, and understand their nature, limitations and performance in Java, you have to know about pointer/address values, pointer variables, getting addresses, dereferencing, etc.

The C/C++ & operator provides the ADDRESS OF its operand, which must be a variable.
(2) of those variables it can point to.

(1) that its a pointer variable, and

The declaration of a pointer variable specifies BOTH

derereferencing operator-hmm...

which, by coincidence, is also the C++
an asterisk

(point to)

(the type of data it can

hold the address of

that the pointer variable

that the pointer variable

The name of the newly

declared pointer

declared pointer

variable

variable

float

* MYFIRSTPOINTER;

Learn how C++ pointer variables are declared.

Important for USING pointers!

©S. Chahken2006
MyP does NOT point to 2003!

Now we can say "MyP POINTS TO MyInt".

```
MyP = &MyInt;
MyInt = 2003;
```

The value of MyP is the address of the int variable MyInt

MEANS

point to variable MyP POINTS TO int variable MyInt
General problem:

has two different names (different kind).

Huh? One variable has two different names (different kinds).

That means: Access the variable whose address is in the pointer (or is the address value).

The * operator dereferences a pointer (or address value).

Important for using pointers!
After Multiplication:

Before Multiplication:

\[
\text{MyP} = *\text{MyP} \times 2;
\]

\[
\text{count} \gg \text{MyP} \gg \text{end}
\]

\[
\text{count} \gg \text{MyP} \gg \text{end}
\]

\[
\text{count} \gg \text{MyInt} \gg \text{end}
\]

\[
\text{count} \gg \text{MyP} \gg \text{end}
\]

\[
\text{count} \gg \text{MyInt} \gg \text{end}
\]

\[
\text{count} \gg \text{MyP} \gg \text{end}
\]

\[
\text{count} \gg \text{MyInt} \gg \text{end}
\]
so it could print the int value located there.

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

Segmentation fault

cout == *MyP  ++ endi;

cout == 0x00000004

cout == " " "  "MyInt  ++ endi;

cout == 4006

cout == MyInt  ++ endi;

After Assignment:

<table>
<thead>
<tr>
<th>4006</th>
<th>0x0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyInt</td>
<td>NULL</td>
</tr>
</tbody>
</table>

MyP = NULL;

Before Assignment:

<table>
<thead>
<tr>
<th>4006</th>
</tr>
</thead>
<tbody>
<tr>
<td>MyInt</td>
</tr>
</tbody>
</table>

MyP | 0x0000004 |

C. Chasen 2006
```cpp
try \cdot \text{cxx} : ? \text{non-trivial in unary}
\text{try} \cdot \text{cxx} : \text{in function, } \text{int main}()

\text{try} \cdot \text{cxx} \text{ \$ \text{\$ \text{\$ \text{\$}

------------------

{ 

cout >> \text{\$ \text{x+3) \text{\$ \text{\$ \text{\$}

\text{int} \text{\$ \text{x = 3; \text{\$ \text{\$ \text{\$}

} 

\text{main} ()

using namespace std; 

#include <iostream>

Expressions like \text{(x+3)} are not variables, so:
```
A variable can store a pointer value.
the 3 variables, before and after each step...

Please simulate the execution of this by writing the values currently in each of

\[ \text{TMP} = \text{P2}^* \]
\[ \text{P2}^* = \text{P1} \]
\[ \text{P1}^* = \text{P1} \]
\[ \text{int} \ \text{TMP}^* \]
\[ \text{int} \ \text{P2}^* \]
\[ \text{int} \ \text{P1}^* \]
\[ \text{int} \ \text{P1}^* \]
\[ \text{int} \ \text{P2}^* \]

How can you program swapping the values of I and J using pointers?

\[ I = 1928; \ J = 2003; \]
\[ \text{int} \ I; \ \text{int} \ J; \]
assert (p1==a) (p2==b)

1928 2003

cout >> i >> j >> end;

2003 1928

cout >> *p1 >> *p2 >> end;

p2 = TEMpp;
pi = p2;
TEMpp = *p1;
int *TEMpp;
p1 = &i;
p2 = &j;
int *p1; int *p2;
int ; int j; i = 1928; j = 2003;

Now, how is swapping the values of the pointer variables different?
What is a `pointer value` (synonyms: address, locator, "Reference" in Java)

What is a `variable` (synonyms: object, memory location, "cell", box for

...
equivelent, \( \ast_{\text{member}}() \) \n\n\text{\texttt{mything}} \rightarrow \text{\texttt{member}}() \); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{copy a ++ pointer}} \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}(); \n\n\text{\texttt{mything}} = \text{\texttt{new Thing}}();
only except you cannot convert them to/from integers.

(2) Java "Reference type" which are like C/C++ pointers (to class objects)

(3) Pointer type (pointer to point; int *point)

(2) Structure/class type (structure of pointable; int, char, float, etc.)

(1) Primitive type (void, float, int, etc.)

Java has only 2:

(1) Primitive type has 3 kinds of named variable and array types:

C/C++ has:

(1) C/C++ assemblymachine language EXPOSE numerical

Java HIDES numerical pointer values/addresses and provides variables values/addresses.

called Java reference variables, to hold and copy Java reference values.

© S. Chakken 2006 University at Albany Computer Science Dept.