Lecture P8: Pointers and Linked Lists

Lewis Caroll
Through the Looking Glass

"The name of the song is called 'Haddocks' Eyes.' "

"Oh, that's the name of the song, is it?" Alice said, trying to feel interested.

"No, you don't understand," the Knight said, looking a little vexed. "That's what the name is called. The name really is 'The Aged Aged Man.' "

"Then I ought to have said 'That's what the song is called' ?" Alice corrected herself.

"No, you oughtn't: that's quite another thing! The song is called 'Ways and Means,' but that is only what it's called, you know!"

"Well, what is the song, then?" said Alice, who was by this time completely bewildered.

"I was coming to that," the Knight said. "The song really is 'A-sitting On A Gate,' and the tune's my own invention."

Pointer Overview

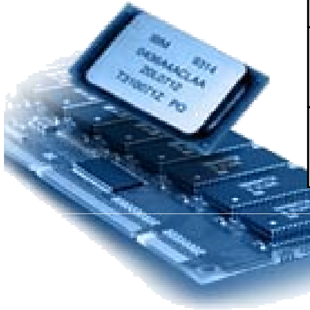
Basic computer memory abstraction.

- Indexed sequence of bits.
- Address = index.
- Ex 1: TOY.
  - basic unit = word = 16 bits
  - 8-bit address refers to one of 256 words
- Ex 2: Arizona.
  - basic unit = byte = 8 bits
  - 32-bit address refers to one of 4 billion+ bytes

addr	value
00	0000
01	3412
02	11AC
03	F00D
04	FADE
05	60B3
06	982A
. . .	. . .
FB	D1CE
FC	CAFE
FD	FECE
FE	CEDE
FF	FACE

Pointer = VARIABLE that holds memory address.

- Allow function to change inputs.
- Create self-referential data structures.
- Better understanding of arrays.



Pointers in TOY

Variable that stores the value of a single MEMORY ADDRESS.

- **In TOY, memory addresses are 00 – FF.**
  - **indirect addressing: store a memory address in a register**
- **Very powerful and useful programming mechanism.**
  - **more confusing in C than in TOY**
  - **easy to abuse!**

Address	D0	D1	D2	..	D9	DA	DB	..	E5	E6	E7
Value	1	9	E5	..	7	0	00	..	3	5	D9

Memory location D2 stores a "pointer" to another memory location (E5) of interest.

Pointer Intuition

Pointer abstraction captures distinction between a thing and its name.

Thing	Name
Web page	www.princeton.edu
Email inbox	august@cs.princeton.edu
This room	Frist 302
Bank account	45-234-23310076
Princeton student	610080478
Word of TOY memory	1A
Byte of PC memory	FFBEFB24
int x;	&x
*px	px declared as int *px;

Pointers in C

C pointers.

- If x is an integer:

  &x is a pointer to x (memory address of x)

- If px is a pointer to an integer:

  *px is the integer

allocate storage for pointer to int

pointer.c

```
#include <stdio.h>

int main(void) {
  int x;
  int *px;

  x = 7;
  px = &x;
  printf("  x = %d\n",   x);
  printf(" px = %p\n",  px);
  printf("*px = %d\n", *px);
  return 0;
}
```

Pointers as Arguments to Functions

Goal: function that swaps values of two integers.

A first attempt:

badswap.c

```
#include <stdio.h>

void swap(int a, int b) {
  int t;
  t = a; a = b; b = t;
}

int main(void) {
  int x = 7, y = 10;
  swap(x, y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Pointers as Arguments to Functions

Goal: function that swaps values of two integers.

Now, one that works.

changes value stored in memory address for x and y

swap.c

```
#include <stdio.h>

void swap(int *pa, int *pb) {
  int t;
  t = *pa; *pa = *pb; *pb = t;
}

int main(void) {
  int x = 7, y = 10;
  swap(&x, &y);
  printf("%d %d\n", x, y);
  return 0;
}
```

Linked List Overview

Goal: deal with large amounts of data.

- Organize data so that it is easy to manipulate.
- Time and space efficient.

Basic computer memory abstraction.

- Indexed sequence of bits (words, bytes).
- Address = index.

Need higher level abstractions to bridge gap.

- Array.
- Struct.
- LINKED LIST
- Binary tree.
- Database.
- . . .

addr	value
00	0000
01	3412
02	11AC
03	F00D
04	FADE
05	60B3
06	982A
. . .	. . .
FB	D1CE
FC	CAFE
FD	DEAF
FE	CEDE
FF	FACE

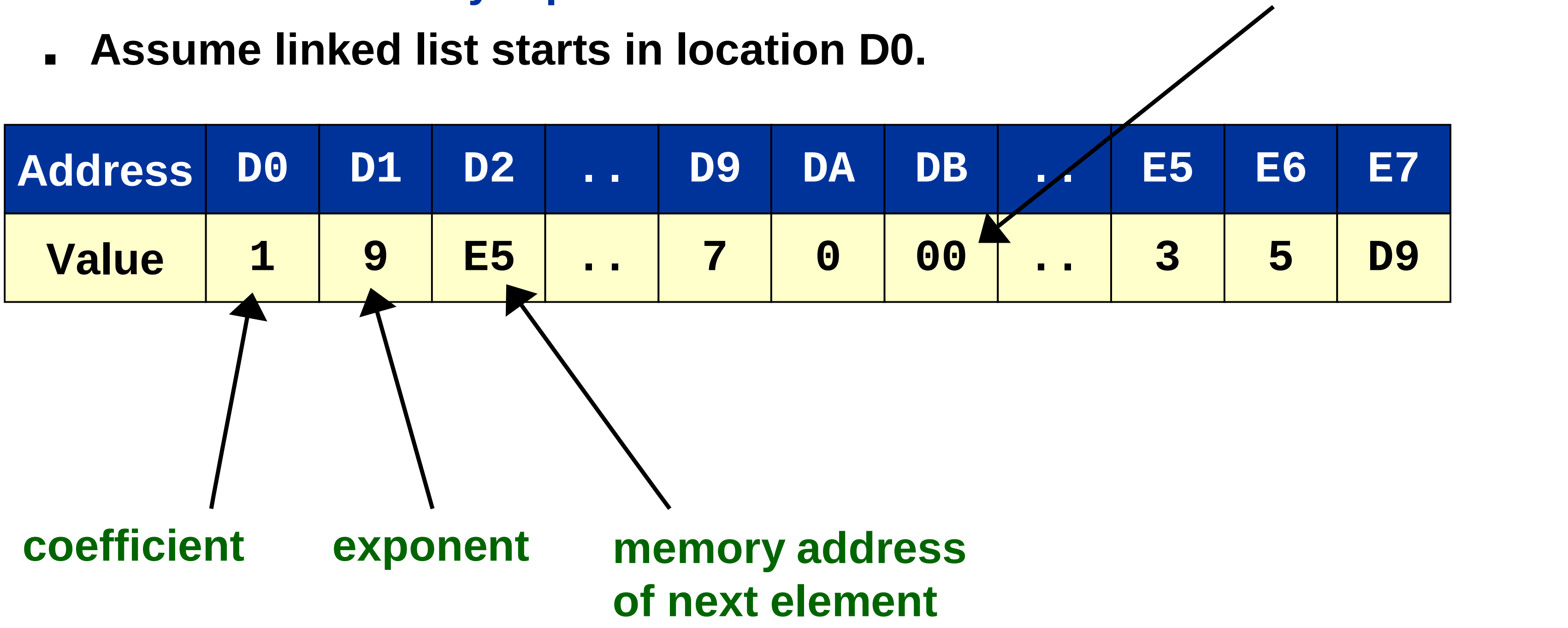
Linked List

Fundamental data structure.

- HOMOGENEOUS collection of values (all same type).
- Store values ANYWHERE in memory.
- Associate LINK with each value.
- Use link for immediate access to the NEXT value.

Possible TOY memory representation of $x^9 + 3x^5 + 7$.

- Assume linked list starts in location D0.



Address	D0	D1	D2	..	D9	DA	DB	..	E5	E6	E7
Value	1	9	E5	..	7	0	00	..	3	5	D9

special "NULL"
memory address
denotes end of list

coefficient

exponent

memory address
of next element

Linked List

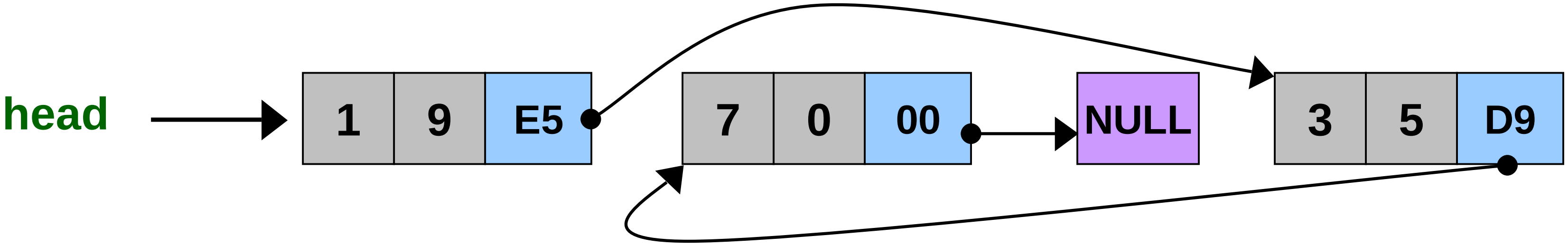
Fundamental data structure.

- HOMOGENEOUS collection of values (all same type).
- Store values ANYWHERE in memory.
- Associate LINK with each value.
- Use link for immediate access to the NEXT value.

Possible TOY memory representation of $x^9 + 3x^5 + 7$.

- Assume linked list starts in location D0.

Address	D0	D1	D2	..	D9	DA	DB	..	E5	E6	E7
Value	1	9	E5	..	7	0	00	..	3	5	D9



Linked List

Fundamental data structure.

- HOMOGENEOUS collection of values (all same type).
- Store values ANYWHERE in memory.
- Associate LINK with each value.
- Use link for immediate access to the NEXT value.

Possible TOY memory representation of $x^9 + 3x^5 + 7$.

- Assume linked list starts in location D0.

Address	D0	D1	D2	..	D9	DA	DB	..	E5	E6	E7
Value	1	9	E5	..	7	0	00	..	3	5	D9

Linked List vs. Array

Polynomial example illustrates basic tradeoffs.

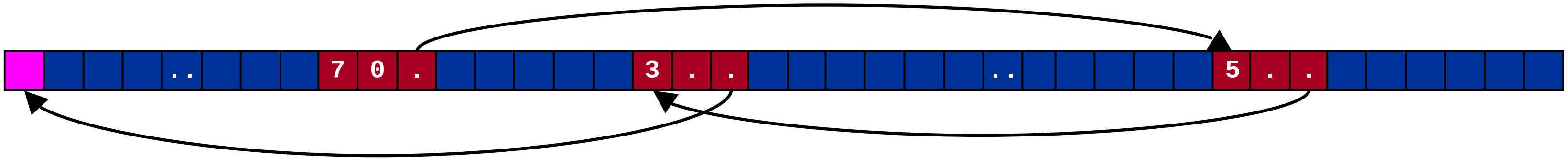
- Sparse polynomial = few terms, large exponent.

  Ex. $3x^{1000000} + 5x^{50000} + 7$

array

7	0	0	0	..	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	..	0	0	0	0	0	0	0	0	0	0	0	0	0	3

linked list



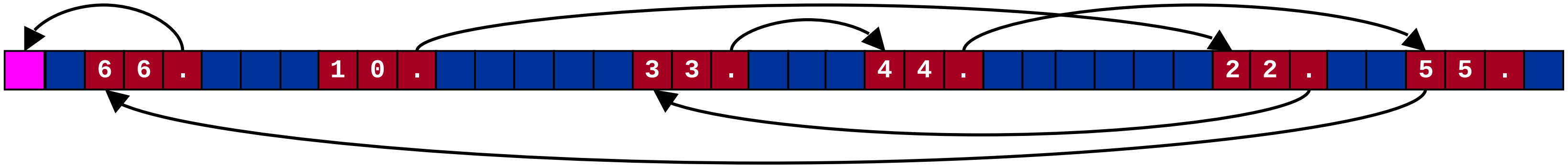
- Dense polynomial = mostly nonzero coefficients.

  Ex. $6x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + 1$

array

1	0	2	3	4	5	6																																	

linked list



Linked List vs. Array

Polynomial example illustrates basic tradeoffs.

Huge Sparse Polynomial		
	array	linked
space	huge	tiny
time	instant	tiny

Huge Dense Polynomial		
	array	linked
space	huge	3 * huge
time	instant	huge

Time to determine coefficient of x^k.

Lesson: know space and time costs.

- Axiom 1: there is never enough space.
- Axiom 2: there is never enough time.

Overview of Linked Lists in C

Not directly built into C language. Need to know:

How to associate pieces of information.

- User-define type using `struct`.
- Include `struct` field for coefficient and exponent.

How to specify links.

- Include `struct` field for POINTER to next linked list element.

How to reserve memory to be used.

- Allocate memory DYNAMICALLY (as you need it).
- `malloc()`

How to use links to access information.

- `->` and `.` operators

Linked List for Polynomial

C code to represent $x^9 + 3x^5 + 7$.

- Statically, using nodes.

poly1.c

```c
struct node {
   int coef;
   int exp;
   struct node *next;
};

int main(void) {
   struct node p, q, r;
   p.coef = 1; p.exp = 9;
   q.coef = 3; q.exp = 5;
   r.coef = 7; r.exp = 0;

   p.next = &q;
   q.next = &r;
   r.next = NULL;
   return 0;
}
```

define node to store 2 integers

memory address of next node

initialize data

link up nodes

- Need to know how many ahead of time.

Linked List for Polynomial

C code to represent $x^9 + 3x^5 + 7$.

- Statically, using nodes.
- Dynamically, using links.

`x->exp` ⇔ `(*x).exp`

initialize data

allocate enough memory to store node

link up nodes of list

Study this code: tip of iceberg!

poly2.c

```
#include <stdlib.h>

typedef struct node *link;
struct node { . . .};

int main(void) {
  link x, y, z;

  x = malloc(sizeof *x);
  x->coef = 1; x->exp = 9;
  y = malloc(sizeof *y);
  y->coef = 3; y->exp = 5;
  z = malloc(sizeof *z);
  z->coef = 7; z->exp = 0;

  x->next = y;
  y->next = z;
  z->next = NULL;

  return 0;
}
```

Review of Stack ADT

Create ADT for stack.

- Lecture P5: implement using an array.
- Now: re-implement using linked list.

STACK.h

```
void STACKinit(void);
int  STACKisempty(void);
void STACKpush(int item);
int  STACKpop(void);
void STACKshow(void);
```

client uses data type, without regard to how it is represented or implemented.

client.c

```
#include "STACK.h"

int main(void) {
  int a, b;
  . . .
  STACKinit();
  STACKpush(a);
  . . .
  b = STACKpop();
  return 0;
}
```

Stack Implementation With Linked Lists

stacklist.c

```
#include <stdlib.h>
#include "STACK.h"

typedef struct STACKnode* link;
struct STACKnode {
   int item;
   link next;
};

static link head;

void STACKinit(void) {
   head = NULL;
}

int STACKisempty(void) {
   return head == NULL;
}
```

standard linked list data structure

`static` to make it a true ADT

head points to top node on stack

Stack Implementation With Linked Lists

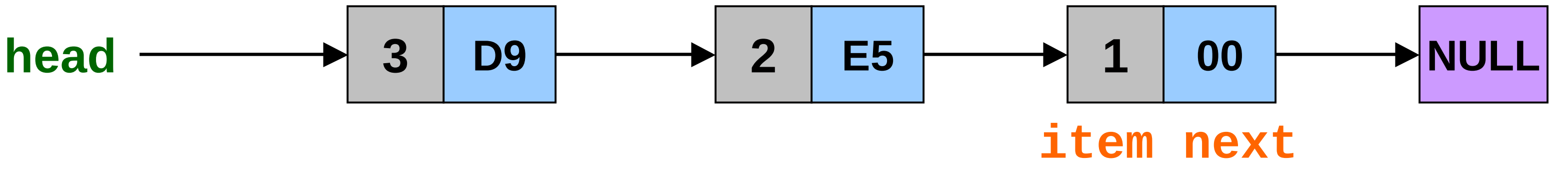
allocate memory and initialize new node

check if malloc fails

insert at beginning of list

stacklist.c (cont)

```
link NEWnode(int item, link next) {
   link x = malloc(sizeof *x);
   if (x == NULL) {
      printf("Out of memory.\n");
      exit(EXIT_FAILURE);
   }
   x->item = item; x->next = next;
   return x;
}
void STACKpush(int item) {
   head = NEWnode(item, head);
}
```



Stack Implementation With Linked Lists

stacklist.c (cont)

```
int STACKpop(void) {
   int value; link second;
   if (head == NULL) {
      printf("Stack underflow.\n");
      exit(EXIT_FAILURE);
   }
   value  = head->item;
   second = head->next;
   free(head);
   head = second;
   return value;
}

void STACKshow(void) {
   link x;
   for (x = head; x != NULL; x = x->next)
      printf("%d\n", x->item);
}
```

free is opposite of `malloc`: gives memory back to system

traverse linked list

Implementing Stacks: Arrays vs. Linked Lists

We can implement a stack with either array or linked list, and switch implementation without changing interface or client.

```
%gcc client.c stacklist.c
%gcc client.c stackarray.c
```

Which is better for stacks?

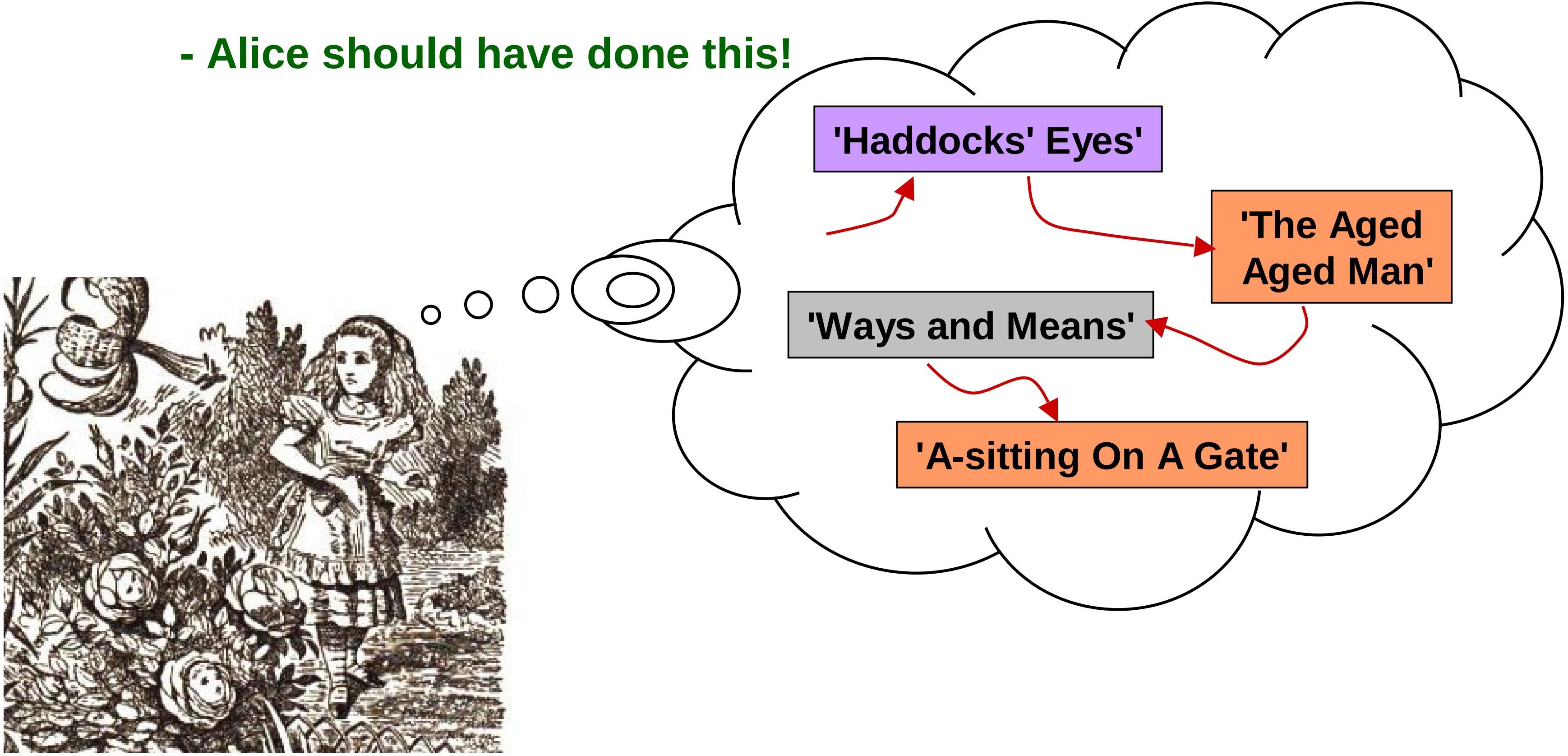
- **Array**
  - ✐
  - ✐
- **Linked List**
  - ✐
  - ✐
  - ✐
  - ✐

Conclusions

Whew, lots of material in this lecture!

- Pointers are useful, but can be confusing. Bewildering, even.
- Study these slides and carefully read relevant material.
- Do not debug by speculatively sprinkling &'s and *'s in your program!
- Instead, do draw pictures with boxes and arrows.

  - Alice should have done this!



'Haddocks' Eyes'

'The Aged Aged Man'

'Ways and Means'

'A-sitting On A Gate'

Lecture P8: Extra Slides

Pointers and Arrays

avg.c

```
#include <stdio.h>
#define N 64

int main(void) {
  int a[N] = {84, 67, 24, ..., 89, 90};
  int i, sum;

  for (i = 0; i < N; i++)
    sum += a[i];

  printf("%d\n", sum / N);
  return 0;
}
```

on arizona,
int is 32 bits (4 bytes) ⇒
4 byte offset

```
"Pointer arithmetic"

&a[0] = a+0   = D000
&a[1] = a+1   = D004
&a[2] = a+2   = D008

 a[0] = *a      = 84
 a[1] = *(a+1) = 67
 a[2] = *(a+2) = 24
```

Memory address	D000	D004	D008	..	D0F8	D0FC	..
Value	84	67	24	..	89	90	..

Pointers and Arrays

Just to stress that `a[i]` really means `*(a+i)`:

```
2[a] = *(2+a) = 24
```

This is legal C, but don't ever do this at home!!!

"Pointer arithmetic"

```
&a[0] = a+0   = D000
&a[1] = a+1   = D004
&a[2] = a+2   = D008

 a[0] = *a      = 84
 a[1] = *(a+1) = 67
 a[2] = *(a+2) = 24
```

Memory address	D000	D004	D008	..	D0F8	D0FC	..
Value	84	67	24	..	89	90	..

Passing Arrays to Functions

Pass array to function.

- Pointer to array element 0 is passed instead.

avg.c

```
#include <stdio.h>
#define N 64

int average(int b[], int n) {
  int i, sum;
  for (i = 0; i < n; i++)
    sum += b[i];
  return sum / n;
}

int main(void) {
  int a[N] = {84, 67, 24, ..., 89, 90};
  printf("%d\n", average(a, N));
  return 0;
}
```

receive the value D000 from main

passes &a[0] = D000 to function

Why Pass Array as Pointer?

Advantages.

- Efficiency for large arrays – don't want to copy entire array.
- Easy to pass "array slice" of "sub-array" to functions.

avg.c

```
int average(int b[], int n) {
  int i, sum;
  for (i = 0; i < n; i++)
    sum += b[i];
  return sum / n;
}

int main(void) {
. . .
  res = average(a+5, 10);
. . .
}
```

compute average of a[5] through a[14]

Passing Arrays to Functions

Many C programmers use `int *b` instead of `int b[]` in function prototype.

- **Emphasizes that array decays to pointer when passed to function.**

average function

```
int average(int b[], int n) {
  int i, sum;
  for (i = 0; i < n; i++)
    sum += b[i];
  return sum / n;
}
```

an equivalent function

```
int average(int *b, int n) {
  int i, sum;
  for (i = 0; i < n; i++)
    sum += b[i];
  return sum / n;
}
```