2D Array, Struct, Malloc

Jinyang Li

based on Tiger Wang’s slides
2D Array

2D arrays are stored contiguously in memory in row-major format.
Multi-dimensional arrays

Declare a k dimensional array

\[
\text{int } \text{arr}[n_1][n_2][n_3]...[n_{k-1}][n_k]
\]

\(n_i\) is the length of the ith dimension
Multi-dimensional arrays

Declare a $k$ dimensional array

```c
int arr[n_1][n_2][n_3]...[n_{k-1}][n_k]
```

$n_i$ is the length of the $i$th dimension

Example: 2D array

```c
int matrix[2][3]
```
Multi-dimensional arrays

Declare a \( k \) dimensional array

\[
\text{int } \text{arr}[n_1][n_2][n_3]...[n_{k-1}][n_k]
\]

\( n_i \) is the length of the \( i \)th dimension

Example: 2D array

\[
\text{int } \text{matrix}[2][3]
\]

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Col 0</th>
<th>Col 1</th>
<th>Col 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Row 1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multi-dimensional arrays

Declare a $k$ dimensional array

\[
\text{int arr}[n_1][n_2][n_3]...[n_{k-1}][n_k]
\]

$n_i$ is the length of the $i$th dimension

Example: 2D array

\[
\text{int matrix}[2][3] = \{\{1, 2, 3\}, \{4, 5, 6\}\};
\]

<table>
<thead>
<tr>
<th>Row 0</th>
<th>Row 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Col 0</th>
<th>Col 1</th>
<th>Col 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Multi-dimensional arrays

Declare a $k$ dimensional array

```ini
int arr[n_1][n_2][n_3]...[n_{k-1}][n_k]
```

$n_i$ is the length of the $i$th dimension

Example: 2D array

```java
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

Access an element at second row and third column

```java
matrix[1][2] = 10
```
Memory layout

```c
int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%p\n", &matrix[i][j]);
    }
}
```
Memory layout

<table>
<thead>
<tr>
<th></th>
<th>0x100</th>
<th>0x104</th>
<th>0x108</th>
<th>0x110</th>
<th>0x114</th>
<th>0x118</th>
<th>0x120</th>
<th>0x124</th>
<th>...</th>
<th>0x400</th>
</tr>
</thead>
<tbody>
<tr>
<td>matrix[0][0]</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix[0][1]</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix[0][2]</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix[1][0]</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix[1][1]</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>matrix[1][2]</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
### Memory layout

<table>
<thead>
<tr>
<th></th>
<th>matrix[1][2]</th>
<th>matrix[1][1]</th>
<th>matrix[1][0]</th>
<th>matrix[0][2]</th>
<th>matrix[0][1]</th>
<th>matrix[0][0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Address</td>
<td>0x10c</td>
<td>0x110</td>
<td>0x114</td>
<td>0x108</td>
<td>0x104</td>
<td>0x100</td>
</tr>
</tbody>
</table>

1st row

0x400
Memory layout

1st row
- matrix[0][0] 1 0x100
- matrix[0][1] 2 0x104
- matrix[0][2] 3 0x108

2nd row
- matrix[1][0] 4 0x10c
- matrix[1][1] 5 0x110
- matrix[1][2] 6 0x114

...
What are the values of matrix, matrix[0] and matrix[1]?

```
int *p1, *p2, *p3;
p1 = (int *)matrix;
p2 = matrix[0];
p3 = matrix[1];
printf("matrix:%p matrix[0]:%p\n matrix[1]:%p
", p1, p2, p3);
```
### Pointers

Matrix:

<table>
<thead>
<tr>
<th>matrix[0][0]</th>
<th>matrix[0][1]</th>
<th>matrix[0][2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0x100</td>
<td>0x104</td>
<td>0x108</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>matrix[1][0]</th>
<th>matrix[1][1]</th>
<th>matrix[1][2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>0x10c</td>
<td>0x110</td>
<td>0x114</td>
</tr>
</tbody>
</table>

- matrix: 0x100
- matrix[0]: 0x100
- matrix[1]: 0x10c

1\(^{st}\) row:
- matrix, matrix[0]

2\(^{nd}\) row:
- matrix[1]

...
Pointers

How many ways to define a pointer which points to the head of the array?
Pointers

```
int *p = &matrix[0][0];
int *p = matrix[0];
int *p = (int *)matrix;
```
Pointers

How to access matrix[1][0] with p?

int *p = &matrix[0][0];
int *p = matrix[0];
int *p = (int *)matrix;
# Pointers

<table>
<thead>
<tr>
<th>matrix[0][0]</th>
<th>matrix[0][1]</th>
<th>matrix[0][2]</th>
<th>matrix[1][0]</th>
<th>matrix[1][1]</th>
<th>matrix[1][2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

```c
int *p = &matrix[0][0];
int *p = matrix[0];
int *p = (int *)matrix;
```

```
matrix[1][0]: *(p + 3)
p[3]
```
A general question

Given a 2D array matrix[m][n] and a pointer p which points to matrix[0][0], how to use p to access matrix[i][j]?
A general question

Given a 2D array matrix\[m\][n] and a pointer \(p\) which points to matrix[0][0], how to use \(p\) to access matrix\[i\][\(j\)]?  

address of matrix\[i\][\(j\)]: \(p + i \times n + j\)
Accessing 2D array using pointer

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 3; j++) {
        printf("%d\n", matrix[i][j]);
    }
}

OR

int *p = matrix[0]; // or int *p = (int *)matrix;
for (int i = 0; i < 2*3; i++) {
    printf("%d\n", p[i]);
}
Structs

Struct stores fields of different types contiguously in memory
Structure

• Array: a block of n consecutive elements of the same type.

• How to define a group of objects, each of which may be of a different type?
struct student {
    int id;
    char name[100];
};
Structure

```c
struct student {
    int id;
    char name[100];
};
```

Field 1: an integer
struct student {
    int id;
    char name[100];
};
Structure

```c
struct student {
    int id;
    char name[100];
};

struct student t; // define an object with type student
```
struct student {
    int id;
    char name[100];
};

struct student t;

t.id = 1024

Access the fields of this object

t.name[0] = ‘z’
t.name[1] = ‘h’
...

**Structure**

```c
typedef struct {
    int id;
    char name[100];
} student;

struct student t;
student *p = &t;

p->id = 1023; // (*p).id = 1023
```

```c
...  
```
typedef struct {
    int id;
    char name[100];
} student;
1\textsuperscript{st} question:
What is the size of structure student?

typedef struct {
    int id;
    char name[100];
} student;
What is the size of structure A?

typedef struct {
    int id;
} A;
What is the size of structure A?

typedef struct {
    int id;
} A;

Answer: 4
What is the size of structure B?

typedef struct {
    char name[100];
} B;
What is the size of structure B?

typedef struct {
    char name[100];
} B;

Answer: 100
1st question:
What is the size of structure student?

typedef struct {
    int id;
    char name[100];
} student;
1st question:
What is the size of structure student?

typedef struct {
    int id;
    char name[100];
} student;

Answer: 104
2\textsuperscript{st} question:
What is the size of structure student?

typedef struct {
    int id;
    char gender;
} student;
2\textsuperscript{st} question:
What is the size of structure student?

```c
typedef struct {
    int id;
    char gender;
} student;
```

\textbf{Answer: 5 ?}
Structure’s size

2nd question:
What is the size of structure student?

typedef struct {
    int id;
    char gender;
} student;

Answer: 5 ?
2\textsuperscript{st} question:
What is the size of structure student?

typedef struct {
    int id;
    char gender;
} student;

Answer: 8
typedef struct {
   int id;
   char gender;
} student;

Memory layout

Structure’s size

one byte

8 bytes

address

Memory layout
typedef struct {
    int id;
    char gender;
} student;
typedef struct {
    int id;
    char gender;
} student;
Put the data at a memory address equal to some multiple of the word size through the data structure padding
Data alignment

Put the data at a memory address equal to some multiple of the primary datatype size through padding

CPU reads/writes data from/into memory in word sized chunks.
(e.g., 8 bytes chunks on a 64-bit system)

Ensure read/write each primary type with a single memory access.
Data alignment

Put the data at a memory address equal to some multiple of the word size through the data structure padding.

```java
readInfo(Student s) {
    int id = s.id
    char gender = s.gender
}

32 bit machine
```
Data alignment

Put the data at a memory address equal to some multiple of the word size through the data structure padding

```
readInfo(student s) {
  int id = s.id
  char gender = s.gender
}
```

32 bit machine
Problem without data alignment

student s[2];

for(int i = 0; i < 2; i++) {
    readInfo(s[i])
}
student s[2];
for(int i = 0; i < 2; i++) {
    readInfo(s[i])
}

readInfo(student s) {
    s[1] → int id = s.id
    char gender = s.gender
}

Problem without data alignment

Memory Layout
student s[2];

for(int i = 0; i < 2; i++) {
    readInfo(s[i])
}

readInfo(student s) {
    int id = s.id
    char gender = s.gender
}

Memory Layout
Problem without data alignment

```
student s[2];
for(int i = 0; i < 2; i++) {
    readInfo(s[i])
}
readInfo(student s) {
    int id = s.id
    char gender = s.gender
}
```

Performance and correctness issues
student s[2];

for(int i = 0; i < 2; i++) {
    readInfo(s[i])
}

readInfo(student s) {
    s[1] → int id = s.id
    char gender = s.gender
}

Memory Layout

...  
...  

S[0]

id  
0x0004
0x0005
0x0006
0x0007
0x0008
0x0009
0x000a
0x000b
0x000c
0x000d

S[1]

padding  
0x0000b
0x0000a
0x00009
0x00008
0x00007
0x00006
0x00005
0x00004

gender  
0x000d
0x000c
0x000b
0x000a
0x0009
0x0008
0x0007
0x0006

id  
0x0000d
0x0000c

...  
...  

...  
...  

memory bus

CPU
Question: how to pad?

typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

typedef struct {
    int a;
    int b;
    char c;
    char d;
} S_B;

Alignment rule:
1. Address of each field f must be multiple of the primary type of f
2. Address of the struct must be multiples of the biggest primary type of all its fields.
   (this ensures a field’s primary data type can be transferred in a single read)
typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

1 word

0x0 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 0x9 0xa 0xb 0xc 0xd 0xe 0xf
typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

1 word

0x0  0x1  0x2  0x3  0x4  0x5  0x6  0x7  0x8  0x9  0xa  0xb  0xc  0xd  0xe  0xf
Questions

typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

1 word
typedef struct {
  int a;
  char b;
  int c;
  char d;
} S_A;
typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

1 word
typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

1 word
typedef struct {
    int a;
    char b;
    int c;
    char d;
} S_A;

1 word
typedef struct {
    int a;
    int b;
    char c;
    char d;
} S_A;

1 word

<table>
<thead>
<tr>
<th>a</th>
<th>0x0</th>
<th>0x1</th>
<th>0x2</th>
<th>0x3</th>
<th>0x4</th>
<th>0x5</th>
<th>0x6</th>
<th>0x7</th>
<th>0x8</th>
<th>0x9</th>
<th>0xa</th>
<th>0xb</th>
<th>0xc</th>
<th>0xd</th>
<th>0xe</th>
<th>0xf</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```c
typedef struct {
    int a;
    int b;
    char c;
    char d;
} S_A;
```

1 word

```
a b
0x0 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 0x9 0xa 0xb 0xc 0xd 0xe 0xf
```
typedef struct {
    int a;
    int b;
    char c;
    char d;
} S_A;

1 word
typedef struct {
    int a;
    int b;
    char c;
    char d;
} S_A;

1 word
typedef struct {
    int id;
    char gender;
} student;

student t = student{1, 'm'};
student *p = &t;
p->id = 2;
Malloocs

Allocates a chunk of memory dynamically
int a[10];

• Global variables are allocated space before program execution.

• Local variables are allocated at the entrance of a function (or a block) and de-allocated upon the exit of the function (or the block)
Malloc

Dynamically allocate a space
- `malloc`: allocate storage of a given size
- `free`: de-allocate previously malloc-ed storage

```c
void *malloc(size_t size);
```

A void pointer is a pointer that has no associated data type with it. A void pointer can hold address of any type and can be casted to any type.

```c
void free(void *ptr);
```
Malloc

Dynamically allocate a space
- malloc: allocate storage of a given size
- free: de-allocate previously malloc-ed storage

```c
#include <stdlib.h>

int *newArr(int n) {
    int *p;
    p = (int*)malloc(sizeof(int) * n);
    return p;
}
```
typedef struct {
    int val;
    struct node *next;
}node;

// insert val into linked list to the head // of the linked list and return the new // head of the list.
node*
insert(node *head, int val) {
}

int main() {
    node *head = NULL;
    for (int i = 0; i < 3; i++)
        head = insert(head, i);
}
Inserting into a linked list

head
Inserting into a linked list

```c
node *insert(node *head, int val) {
    node *n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = head;
}
```
Inserting into a linked list

```
node *insert(node *head, int val) {
    node *n = (node *)malloc(sizeof(node));
    n->val = val;
    n->next = head;
    return n;
}
```
Exercise 1: Reverse a linked list

```c
struct node {
    int val;
    struct node *next;
};

struct node* reverseList(struct node* head) {
    // your code here
}
```
Reverse a linked list

head

prev
cur

next
Reverse a linked list

cur->next = prev
Reverse a linked list

```
cur->next = prev
prev = cur
```
Reverse a linked list

head

prev

cur

next

cur->next = prev
prev = cur
cur = next
Reverse a linked list

- head
- prev
- cur
- next

- cur->next = prev
- prev = cur
- cur = next
- next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

```
cur->next = prev
prev = cur
cur = next
next = cur->next
```
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

cur->next = prev
prev = cur
cur = next
next = cur->next
Reverse a linked list

```c
struct node {
    int val;
    struct node *next;
};

struct node*
reverseList(struct node* head) {
    node *prev = null;
    node *curr = head;
    while (curr != null) {
        node *next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}
```
Exercise 2: Remove an element

struct node {
    int val;
    struct node *next;
};

struct node*
removeElements(struct node* head, int val)
{
    // your code here
}

Example
Given: 1 → 2 → 6 → 3 → 6 → 8, val = 6
Return: 1 → 2 → 3 → 8
Remove linked list element

check prev->next->val
Remove linked list element

But how to remove the first element?

```cpp
check prev->next->val
if prev->next->val == val {
    prev->next->next = prev->next
}
```
Remove linked list element

Basic idea: add a fake node at beginning
struct node {
    int val;
    struct node *next;
};

struct node*
removeElements(struct node* head, int val) {
    struct node *n = (struct node *)malloc(sizeof(struct node));
    struct node *r = n;

    n->next = head;
    while(n->next != NULL) {
        if (n->next->val == val) {
            n->next = n->next->next;
        } else {
            n = n->next;
        }
    }

    return r->next;
}