Structures

2018 Fall
Joonwon Lee
Review

- **Dynamic Memory Allocation**

char str1[] = "hello";

char *str2 = (char*) malloc(sizeof(char) * 6);
strcpy(str2, str1);

Unchangeable

```
str1
```

Changeable

```
str2
```

![Diagram showing the allocation and copying of strings str1 and str2](image-url)
Review

- Double Pointer

```c
char **strings;

strings = (char **)malloc(sizeof(char *) * n);
```

```
0x1234
```

```
[0]
[1]
[2]
...[n-1]
```

```
strings
```

```
hello
world
bye
lab
```
Structure vs Union

```c
struct str {
    int num;
    unsigned char arr[4];
};

union uni {
    int num;
    unsigned char arr[4];
};
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int num</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

int num

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
struct person1 {
    char firstName[20];
    char lastName[20];
    int age;
};

struct person1
Structures

```c
struct person1 p1 = {"michael", "jackson", 20};

typedef struct person1 P;

P p1 = {"michael", "jackson", 20};

P p2;
```

<table>
<thead>
<tr>
<th>m</th>
<th>i</th>
<th>c</th>
<th>h</th>
<th>a</th>
<th>e</th>
<th>l</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>a</td>
<td>c</td>
<td>k</td>
<td>s</td>
<td>o</td>
<td>n</td>
<td>20</td>
</tr>
</tbody>
</table>

p1
struct person2 {
    char *firstName;
    char *lastName;
    int age;
};

struct person2

[Diagram of struct person2 with arrows pointing to the members]

GEDB029 Basis and Practice in Programming | Fall 2018
struct person2 p2 = {"michael", "jackson", 20};

p2.firstName = (char *)malloc(...);
p2.lastName = (char *)malloc(...);
strcpy(p2.firstName, "michael");
strcpy(p2.lastName, "jackson");
Structures

- Call-by-Value

```c
void func1(struct person1 p)
{
    strcpy(p.firstName, "stevie");
    strcpy(p.lastName, "wonder");
    p.age = 50;
}
```
Structures

- Call-by-Reference

```c
void func1(struct person1 *p)
{
    strcpy(p->firstName, "stevie");
    strcpy(p->lastName, "wonder");
    p->age = 50;
}
```

- \((\ast p).\text{field} == p->\text{field}\)
Linked-List

2018 Fall
Joonwon Lee
Linked-List

- **Overview**

- A linked list is a series of connected nodes
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- Head: pointer to the first node
- The last node points to *NULL*
struct Link {
    int data;
    struct Link *next;
};

struct Link *head, *tail;
struct Link {
    int data;
    struct Link *next;
};

struct Link *head;

InsertNode(...);
struct Link {
    int data;
    struct Link *next;
};

struct Link *head;

DeleteNode(...);

head

curNode

prevNode
curNode
Linked-List

- **Linked-List vs Array**
  - Linked lists are complex to code and manage.
  - Some distinct advantages
    - Dynamic: a linked list can easily grow and shrink in size.
    - Easy and fast insertions and deletions

- **External Resources**
  - [https://opentutorials.org/module/1335/8821](https://opentutorials.org/module/1335/8821)