

ساختار های پیچیده داده

Start Slide



ستاره کاشانچی

روزین صداقت

ابوالفضل صادق نژاد

امیر مهدی علی نوری

مبانی و ضرورت
استقاده از ساختار

01

نحوه‌ی تعریف، مقداردهی و
طراحی ساختارها

02

کار با آرایه‌ها و توابع
در کنار ساختارها

03

اشاره‌گرها و مدیریت
پویا در ساختارها

04

کاربردهای پیشرفته و
حرفه‌ای ساختارها

۰۵

بخش اول : مبانی و ضرورت استفاده از ساختارها در زبان C

تعریف ساختار (Structure) در زبان C

یک نوع داده‌ی تعریف شده توسط کاربر است

```
c

#include <stdio.h>

struct Student {
    int id;
    float avg;
};

int main() {
    struct Student s1;
    s1.id = 1;
    s1.avg = 18.5;

    printf("ID: %d\n", s1.id);
    printf("Average: %.2f\n", s1.avg);

    return 0;
}
```

شامل چند متغیر مرتبط

با نوع‌های داده‌ی متفاوت

چرا structure لازم است؟

کاهش تعداد متغیر ها

نظم دهی داده ها

افزایش خوانایی

با ساختار : 

 بدون ساختار :

c

```
struct Student s1, s2;
```

c

```
int id1, id2;  
float avg1, avg2;
```

تفاوت با آرایه ها



ساختار:

داده های غیر هم نوع

آرایه:

داده های هم نوع

```
c  
  
struct Student {  
    char name[20];  
    int id;  
    float avg;  
};
```

```
c  
  
int scores[3] = {18, 17, 19};
```

مدلسازی دنیای واقعی

مثال ها : دانشجو --- کتاب --- محصول

```
c

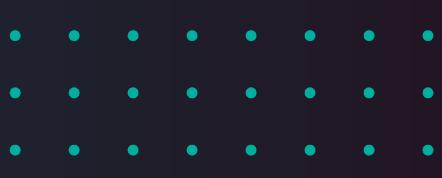
struct Book {
    char title[50];
    int year;
    float price;
};
```



ساخت چند نمونه از یک Structure

امکان ایجاد چند نمونه که هر کدام از آنها داده‌ای مستقل از یکدیگرند.

```
c  
  
struct Student s1 = {1, 18.2};  
struct Student s2 = {2, 16.9};
```



جمع بندی بخش اول

[Home](#)[About](#)[Resources](#)

```
c

#include <stdio.h>

struct Product {
    int code;
    float price;
};

int main() {
    struct Product p;
    p.code = 101;
    p.price = 250.75;

    printf("Code: %d\n", p.code);
    printf("Price: %.2f\n", p.price);

    return 0;
}
```



بخش دوم: نحوه تعریف ، مقداردهی و طراحی ساختار ها

```
c  
#include <stdio.h>  
  
struct User {  
    int id;  
    int age;  
};  
  
int main() {  
    struct User u1;  
    u1.id = 1001;  
    u1.age = 21;  
  
    printf("ID: %d\n", u1.id);  
    printf("Age: %d\n", u1.age);  
  
    return 0;  
}
```

نحوه تعریف یک ساختار و ایجاد نمونه

تعریف ساختار با `struct`

تعیین اعضا

ساخت نمونه از `structure`

دسترسی به اعضای ساختار با عملگر نقطه

دسترسی به اعضا با نقطه (.)
فقط برای متغیر های معمولی structure

```
c

struct User u1;
u1.id = 200;
u1.age = 30;
```

مقداردهی اولیه Structure

مقداردهی هنگام تعریف که در آن ترتیب اعضا مهم است.

c

```
struct User u2 = {101, 25};
```

اگر ترتیب اشتباه وارد بشه ، برنامه خطا نمی ده ولی نتیجه غلط می شه

تغییر مقادیر اعضای Structure

امکان تغییر مقادیر در هر زمان ،
مشابه متغیر های عادی

```
c
```

```
u2.age = 26;
```



اصول طراحی Structure خوانا و منظم

نامگذاری واضح --- ترتیب منطقی اعضا --- جلوگیری از پیچیدگی بی مورد

```
c

struct A {
    int x;
    char b[50];
    float y;
};
```

✖ طراحی بد

```
c

struct Student {
    int id;
    char name[50];
    float avg;
};
```

✓ طراحی خوب

استفاده از `typedef` برای ساده سازی

حذف نوشتگر مکرر `struct`
افزایش خوانایی کد

```
c

typedef struct {
    int id;
    float salary;
} Employee;

int main() {
    Employee e1;
    e1.id = 10;
    e1.salary = 5000.0;

    return 0;
}
```

تفاوت **struct** معمولی با **typedef struct** شده

بدون **typedef** رسمی تر و طولانی تر

با **typedef** ساده تر

```
c
```

```
struct User u;
```

```
c
```

```
User u;
```



مثال عملی : طراحی ساختار مدیریت کاربر

```
c
#include <stdio.h>

typedef struct {
    int id;
    char username[30];
    char password[30];
    int isActive;
} User;

int main() {
    User u1 = {1, "admin", "1234", 1};

    printf("ID: %d\n", u1.id);
    printf("Username: %s\n", u1.username);
    printf("Active: %d\n", u1.isActive);

    return 0;
}
```

طراحی اطلاعات پایه کاربر به صورت
خوانا و قابل توسعه

نکات مهم برای طراحی حرفه ای

حداقل سازی اعضا : بتوان بعدا به آن اعضای جدید اضافه کرد .

آمادگی برای توسعه : وابستگی زیاد ایجاد نکند .

هماهنگی با پروژه های بزرگ : در توابع مختلف قابل استفاده باشد .



جمع بندی بخش دوم

[Home](#)[About](#)[Resources](#)

```
c

#include <stdio.h>

/* تعریف پایه Structure */
typedef struct {
    int id;
    char name[40];
    int age;
    float avg;
} Student;

/* تابع چاپ اطلاعات دانشجو */
void printStudent(Student s) {
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("Age: %d\n", s.age);
    printf("Average: %.2f\n", s.avg);
    printf("-----\n");
}

int main() {
    /* مقدارهای اولیه */
    Student s1 = {101, "Ali Ahmadi", 20, 17.85};

    /* چاپ اولیه */
    printStudent(s1);

    /* تغییر مقدار اعضاء */
    s1.age = 21;
    s1.avg = 18.40;

    /* چاپ بعد از تغییر */
    printStudent(s1);

    return 0;
}
```



ایجاد آرایه از ساختار ها

ذخیره چند داده ی مشابه و مدیریت مجموعه ای از داده ها و اشیای هم نوع

```
c

#include <stdio.h>

typedef struct {
    int id;
    char name[30];
    float avg;
} Student;

int main() {
    Student students[3];

    students[0].id = 1;
    students[1].id = 2;
    students[2].id = 3;

    return 0;
}
```

مقداردهی اولیه آرایه‌ی structure

مقداردهی همزمان چند structure و خوانایی بالاتر آن

c

```
Student students[2] = {  
    {1, "Ali", 18.5},  
    {2, "Sara", 19.2}  
};
```

ارسال (By Value) به تابع structure

```
c  
void printStudent(Student s) {  
    printf("%d %s %.2f\n", s.id, s.name, s.avg);  
}
```

کپی کامل structure فقط برای خواندن اطلاعات مناسب است.

ارسال (By Pointer) به تابع structure

```
c  
void increaseAvg(Student *s) {  
    s->avg += 0.5;  
}
```

بدون کپی کردن و امکان تغییر مقدار

تفاوت By Pointer و By Value

امن ولی پرہزینہ <= By Value

سریع و قابل تغییر <= By Pointer

هر جا تغییر بخوایم ، پوینتر میفرستیم!



پیمايش آرایه ای از

استفاده از حلقه for برای دسترسی به اعضا

```
c  
  
for (int i = 0; i < 2; i++) {  
    printf("%s %.2f\n", students[i].name, students[i].avg);  
}
```

جستجو در آرایه ای از

پیدا کردن داده خاص

استفاده در پروژه ها

C

```
int searchById(Student students[], int n, int id) {  
    for (int i = 0; i < n; i++) {  
        if (students[i].id == id)  
            return i;  
    }  
    return -1;  
}
```

خروجی تابع اندیس جستجو یا `i`- است.

مرتب سازی آرایه‌ی Structure

مرتب سازی بر اساس فیلد خاص --- الگوریتم ساده

```
c

void sortByAvg(Student students[], int n) {
    Student temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (students[i].avg < students[j].avg) {
                temp = students[i];
                students[i] = students[j];
                students[j] = temp;
            }
        }
    }
}
```

در اینجا کل **Structure** جا به جا میشے ، نه فقط یک عضو!

بخش اول : اشاره گر ها و مدیریت پویا در ساختار ها

تعریف اشاره گر به Structure

تعریف اشاره گر به Structure

نگهداری ادرس ساختار در حافظه

```
c  
  
Student s1;  
Student *ptr;  
  
ptr = &s1;
```

مدیریت لیست دانشجویان

```
c

#include <stdio.h>

typedef struct {
    int id;
    char name[30];
    float avg;
} Student;

void printStudents(Student s[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d %s %.2f\n", s[i].id, s[i].name, s[i].avg
}

int main() {
    Student students[3] = {
        {1, "Ali", 17.5},
        {2, "Sara", 19.1},
        {3, "Reza", 16.8}
    };

    printStudents(students, 3);
    return 0;
}
```



تفاوت عملگر . و -<

'-<' برای متغیر Structure

```
c  
ptr->id = 10;
```

'. ' برای متغیر Structure

```
c  
s1.id = 10;
```

-< یعنی اول برو به آدرس و بعدش عضو رو بردار

تخصیص پویای malloc با Structure

ساخت Heap Structure در
اندازه حافظه مشخص

چون در برنامه های واقعی معمولا تعداد Structure از قبل مشخص نبوده و برای این کار از حافظه پویا (Dynamic Memory) استفاده می شود.

```
c  
  
Student *s;  
  
s = (Student *)malloc(sizeof(Student));
```

! Stack تو Heap ساخته شده ، نه

مقداردهی به Structure داینامیک

استفاده از -> برای مقداردهی اعضا

c

```
s->id = 1;  
s->avg = 18.75;
```

اینجا استفاده از . کاملاً غلطه ! 

آزادسازی حافظه (free)

برای جلوگیری از Memory Leak
و مدیریت درست حافظه

```
c  
  
free(s);  
s = NULL;
```

مثال کامل از Structure داینامیک

```
c  
  
#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct {  
    int id;  
    float avg;  
} Student;  
  
int main() {  
    Student *s;  
  
    s = (Student *)malloc(sizeof(Student));  
  
    s->id = 5;  
    s->avg = 19.2;  
  
    printf("%d %.2f\n", s->id, s->avg);  
  
    free(s);  
    return 0;  
}
```

حذف نوشتگان مکرر struct
افزایش خوانایی کد

خود ارجاع (Self-Referential) Structure

اشاره گر به خودش : داخل ساختار ، یک اشاره گر به همان نوع ساختار وجود دارد .

پایه لیست پیوندی ، Tree و Graph

```
c
typedef struct Node {
    int data;
    struct Node *next;
} Node;
```

ساخت یک Node از لیست پیوندی

c

```
Node *n1 = (Node *)malloc(sizeof(Node));  
Node *n2 = (Node *)malloc(sizeof(Node));  
  
n1->data = 10;  
n1->next = n2;  
  
n2->data = 20;  
n2->next = NULL;
```

10 → 20 → NULL

ترکیب + Structure + Pointer آرایه پویا

در پروژه های واقعی چون تعداد داده ها مشخص نیست(نامحدود است) حافظه باید داینامیک باشد که Pointer + Structure انتخاب برای مدیریت آن هاست.

```
c  
  
Student *students;  
int n = 3;  
  
students = (Student *)malloc(n * sizeof(Student));
```

جمع بندی بخش چهارم

[Home](#)[About](#)[Resources](#)

```
c

#include <stdio.h>
#include <stdlib.h>

/* تعریف ساختار خود ارجاع (نود لیست پیوندی) */
typedef struct Node {
    int data;
    struct Node *next;
} Node;

/* تابع ایجاد نود جدید */
Node* createNode(int value) {
    Node *newNode = (Node *)malloc(sizeof(Node));

    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return NULL;
    }

    newNode->data = value;
    newNode->next = NULL;
    return newNode;
}

/* تابع حاپ لیست */
void printList(Node *head) {
    Node *current = head;

    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}

/* تابع آزادسازی کل لیست */
void freeList(Node *head) {
    Node *temp;

    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    /* ایجاد نودها به صورت زیر */
    Node *head = createNode(10);
    Node *second = createNode(20);
    Node *third = createNode(30);

    /* اتصال نودها */
    head->next = second;
    second->next = third;

    /* حاپ لیست */
    printList(head);

    /* آزادسازی حافظه */
    freeList(head);

    return 0;
}
```

کاربرد های پیشرفته و حرفه ای ساختار ها

اصلن چرا کاربرد ها مهم‌ان؟

ورود به سطح حرفه ای

استفاده واقعی از ساختار ها

بررسی حافظه و سیستم

ساختار های تو در تو (Nested Structures)

Structure داخل Structure

مدلسازی داده های چند لایه

```
c

typedef struct {
    char city[30];
    char street[50];
    int postalCode;
} Address;

typedef struct {
    int id;
    char name[30];
    Address address;
} Student;
```

دسترسی به اعضای Nested Structures

```
c  
Student s;  
s.address.postalCode = 45678;
```

دسترسی زنجیره ای

خوانا و منطقی

این خط یعنی « برو داخل Student ، بعد Address ، postalCode »

Nested Structures با Pointer

نرکیب Pointer و Nested و پرکاربرد در پروژه های واقعی

```
c  
Student *sp;  
sp = &s;  
  
sp->address.city[0] = 'T';  
.....
```

استفاده از ساختار در پروژه های بزرگ

جداسازی کد

نگهداری آسان

توسعه پذیری

```
c

// user.h
typedef struct {
    int id;
    char username[30];
} User;
```

ارتباط Structure با فایل ها

ساختار ، بهترین گزینه برای ذخیره داده ها در فایل ها
بازیابی داده ها => مثل یک دیتابیس ساده

نوشتن Structure در فایل باینری

```
c
```

```
FILE *fp = fopen("students.bin", "wb");
fwrite(&s, sizeof(Student), 1, fp);
fclose(fp);
```

این دیتا مستقیما به حافظه دیسک میره .



خواندن از فایل پاینتری Structure

```
c  
  
FILE *fp = fopen("students.bin", "rb");  
fread(&s, sizeof(Student), 1, fp);  
fclose(fp);
```

سریع‌تر از فایل متنی، اما
وابسته به ساختار حافظه

حافظه Structure در زبان C

در حافظه :

اعضای Structure پشت سر هم ذخیره میشوند اما ممکنه فضای خالی اضافه بشود

چرا؟ خطاط Alignment

چیه؟ Alignment

هم ترازی داده ها و افزایش سرعت پردازنده

CPU دوست دارد داده ها در آدرس های خاص هم تراز یا همون Aligned باشند
برای همی کامپایلر بین اعضا Padding اضافه می کند.

Structure Padding

Char = 1 byte

Int = 4 bytes

Structure = 8 bytes

۳ بایت Padding اضافه شده!

```
c
struct Test {
    char a;
    int b;
};
```



بهینه سازی Padding

اهمیت ترتیب اعضاء و کاهش سرعت حافظه

```
c

struct Test {
    int b;
    char a;
};
```

تفاوت Struct با Union

```
c

union Data {
    int i;
    float f;
};
```

حافظه جدا > Struct

حافظه مشترک > Union

اندازه union = اندازه بزرگترین عضو

کاربرد union

صرفه جویی حافظه --- استفاده بیشتر در سیستم های سطح پایین

در Embedded systems ، درایور ها ، و پروتکل ها union بسیار مهم است

کاربرد Structure در سیستم های واقعی

در سیستم عامل ها ، شبکه و مدیریت فایل و ... کاربرد دارند

یه جورایی همه چیز در C با Structure ساخته شده



پروژه نهایی : سیستم مدیریت کتابخانه

طراحی واقعی --- ساختار تو در تو --- آرایه و مدیریت داده

```
c

typedef struct {
    int id;
    char title[50];
    char author[40];
} Book;

typedef struct {
    Book books[100];
    int count;
} Library;
```

تحلیل پروژه کتابخانه

یک مجموعه از Book هاست تعداد کتاب ها رو نگه میداره . count پایه یک سیستم واقعی هست .

[Home](#)
[About](#)
[Resources](#)

```
c

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

#define MAX_BOOKS 100

typedef struct {
    int id;
    char title[50];
    char author[40];
} Book;

typedef struct {
    Book books[MAX_BOOKS];
    int count;
} Library;

/* افزودن کتاب */
void addBook(Library *lib, int id, char title[], char author[]) {
    if (lib->count >= MAX_BOOKS) {
        printf("Library is full!\n");
        return;
    }

    lib->books[lib->count].id = id;
    strcpy(lib->books[lib->count].title, title);
    strcpy(lib->books[lib->count].author, author);
    lib->count++;
}

/* نمایش همه کتابها */
void printBooks(Library lib) {
    for (int i = 0; i < lib.count; i++) {
        printf("ID: %d | Title: %s | Author: %s\n",
               lib.books[i].id,
               lib.books[i].title,
               lib.books[i].author);
    }
}

/* ID جستجو بر اساس */
int searchBookById(Library lib, int id) {
    for (int i = 0; i < lib.count; i++) {
        if (lib.books[i].id == id)
            return i;
    }
    return -1;
}

int main() {
    Library myLibrary;
    myLibrary.count = 0;

    addBook(&myLibrary, 1, "C Programming", "Dennis Ritchie");
    addBook(&myLibrary, 2, "Data Structures", "Mark Weiss");
    addBook(&myLibrary, 3, "Operating Systems", "Silberschatz");

    printBooks(myLibrary);

    int index = searchBookById(myLibrary, 2);
    if (index != -1)
        printf("Book found: %s\n", myLibrary.books[index].title);
    else
        printf("Book not found!\n");

    return 0;
}
```

جمع بندی بخش پنجم

[Home](#)[About](#)[Resources](#)

```
c

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

/* (Nested) ساختار آدرس */
typedef struct {
    char city[30];
    int postalCode;
} Address;

/* ساختار دانشجو */
typedef struct {
    int id;
    char name[30];
    Address addr;
} Student;

/* union برای مقایسه با struct */
union Data {
    int i;
    float f;
};

int main() {
    /* تخصیص پوینت */
    Student *s = (Student *)malloc(sizeof(Student));

    /* مقداردهی */
    s->id = 1;
    strcpy(s->name, "Ali");
    strcpy(s->addr.city, "Tehran");
    s->addr.postalCode = 12345;

    /* اطلاعات چاپ */
    printf("ID: %d\n", s->id);
    printf("Name: %s\n", s->name);
    printf("City: %s\n", s->addr.city);
    printf("Postal Code: %d\n", s->addr.postalCode);

    /* کار با union */
    union Data d;
    d.i = 10;
    printf("Union int: %d\n", d.i);
    d.f = 3.14;
    printf("Union float: %.2f\n", d.f);

    /* ذخیره در فایل باینری */
    FILE *fp = fopen("student.bin", "wb");
    fwrite(s, sizeof(Student), 1, fp);
    fclose(fp);

    /* آزادسازی حافظه */
    free(s);

    return 0;
}
```

جمع‌بندی کل

[Home](#)
[About](#)
[Resources](#)

```
c

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

/* ----- Nested Structure ----- */
typedef struct {
    char city[30];
    int postalCode;
} Address;

/* ----- Main Structure ----- */
typedef struct {
    int id;
    char name[30];
    float avg;
    Address address;
} Student;

/* ----- Self-Referential Structure ----- */
typedef struct Node {
    Student data;
    struct Node *next;
} Node;

/* ----- Union Example ----- */
union Info {
    int code;
    float score;
};

/* ----- Function Prototypes ----- */
Node* createNode(Student s);
void printStudent(Student s);
void freeList(Node *head);

/* ----- Create Linked List Node ----- */
Node* createNode(Student s) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->data = s;
    newNode->next = NULL;
    return newNode;
}
```

```
/* ----- Print Student ----- */
void printStudent(Student s) {
    printf("ID: %d | Name: %s | Avg: %.2f | City: %s\n",
           s.id, s.name, s.avg, s.address.city);
}

/* ----- Free Linked List ----- */
void freeList(Node *head) {
    Node *temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    /* ----- Array of Structures ----- */
    Student students[2] = {
        {1, "Ali", 18.5, {"Tehran", 11111}},
        {2, "Sara", 19.2, {"Shiraz", 22222}}
    };

    /* ----- Linked List ----- */
    Node *head = createNode(students[0]);
    head->next = createNode(students[1]);

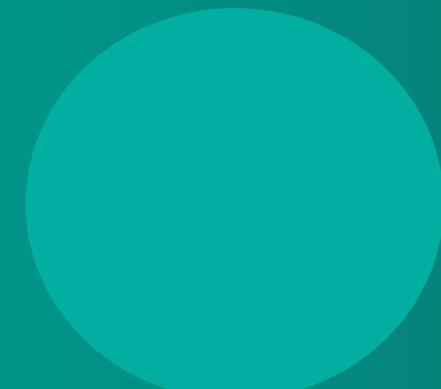
    /* ----- Print List ----- */
    Node *current = head;
    while (current != NULL) {
        printStudent(current->data);
        current = current->next;
    }

    /* ----- File Handling ----- */
    FILE *fp = fopen("students.bin", "wb");
    fwrite(students, sizeof(Student), 2, fp);
    fclose(fp);
}
```

```
/* ----- Union Usage ----- */
union Info info;
info.code = 404;
printf("Union code: %d\n", info.code);
info.score = 17.75;
printf("Union score: %.2f\n", info.score);

/* ----- Memory Cleanup ----- */
freeList(head);

return 0;
```



تمرین ها :

51

55

Home

About

Resources

در یک برنامه شبکه‌ای، لازم است پیامی ارسال شود که در هر لحظه فقط یکی از انواع داده زیر را در خود نگه می‌دارد:

- عدد صحیح
- عدد اعشاری
- یک کاراکتر

برنامه‌ای طراحی کنید که بتواند این پیام را ذخیره کرده، مقداردهی کند و مقدار آن را نمایش دهد، به‌گونه‌ای که مصرف حافظه بهینه باشد.

در یک برنامه مدیریت کتابخانه، اطلاعات کتاب‌ها شامل عنوان، نویسنده، سال انتشار و شماره شناسایی ذخیره می‌شود.

این اطلاعات باید در یک فایل ذخیره شوند تا پس از بسته شدن برنامه، در اجرای بعدی دوباره قابل استفاده باشند.

برنامه‌ای بنویسید که: اطلاعات کتاب‌ها را با استفاده از Structure ذخیره کند داده‌ها را در فایل باینری بنویسد در اجرای بعدی، اطلاعات را از فایل خوانده و نمایش دهد

می‌خواهید یک برنامه برای مدیریت اطلاعات دانشجویان بنویسید که در آن: تعداد دانشجویان از قبل مشخص نیست اطلاعات هر دانشجو شامل شماره دانشجویی، نام و معدل است امکان اضافه کردن دانشجو جدید در زمان اجرا وجود دارد امکان حذف دانشجو از لیست وجود دارد امکان پیمایش و نمایش کل لیست فراهم است

تمام داده‌ها باید به صورت پویا در حافظه ذخیره شوند و پس از پایان برنامه، حافظه به درستی آزاد شود.

در یک پروژه سیستمی، قرار است تعداد زیادی رکورد داده در حافظه ذخیره شود و مصرف حافظه اهمیت بسیار بالایی دارد.

شما باید Structure هایی طراحی کنید که شامل چند نوع داده مختلف (کاراکتر، عدد صحیح و عدد کوتاه) هستند.

هدف این است که: ساختارها کمترین فضای ممکن از حافظه را اشغال کنند طراحی به شکلی انجام شود که Padding اضافی به حداقل برسد

برنامه‌ای بنویسید که اندازه حافظه Structure ها را محاسبه کرده و نمایش دهد.

فرض کنید قرار است یک برنامه به زبان C طراحی کنید که اطلاعات کاربران یک سامانه آموزشی را مدیریت کند.

هر کاربر می‌تواند در چند دوره آموزشی ثبت‌نام کرده باشد و اطلاعات هر دوره باید به صورت مجزا نگهداری شود.

تعداد کاربران و تعداد دوره‌های هر کاربر از قبل مشخص نیست و در زمان اجرای برنامه تعیین می‌شود.

برنامه باید به‌گونه‌ای طراحی شود که امکان: ایجاد کاربران جدید ذخیره اطلاعات دوره‌های مربوط به هر کاربر نمایش کامل اطلاعات هر کاربر همراه با دوره‌هایش را داشته باشد.

(پیاده‌سازی باید مبتنی بر Structure، اشاره‌گر و مدیریت حافظه پویا باشد.)

نمایش پاسخ



نمایش پاسخ



نمایش پاسخ



نمایش پاسخ



طراحی پیام شبکه‌ای

ذخیره سازی اطلاعات کتابخانه

مدیریت لیست پویای دانشجویان

بهینه سازی حافظه ساختارها

سیستم مدیریت کاربران آموزشی

پاسخ تمرین ها :

52

55

Home

About

Resources

```
C

#include <stdio.h>

union Message {
    int intValue;
    float floatValue;
    char charValue;
};

int main() {
    union Message msg;

    msg.intValue = 100;
    printf("Int: %d\n", msg.intValue);

    msg.floatValue = 3.14;
    printf("Float: %.2f\n", msg.floatValue);

    msg.charValue = 'A';
    printf("Char: %c\n", msg.charValue);

    return 0;
}
```

```
C

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

typedef struct {
    int id;
    char name[50];
    float gpa;
    struct Student *next;
} Student;

Student* createStudent(int id, char name[], float gpa) {
    Student* newStudent = (Student*)malloc(sizeof(Student));
    newStudent->id = id;
    strcpy(newStudent->name, name);
    newStudent->gpa = gpa;
    newStudent->next = NULL;
    return newStudent;
}

void printStudents(Student *head) {
    Student *current = head;
    while(current != NULL) {
        printf("ID: %d | Name: %s | GPA: %.2f\n", current->id, current->name, current->gpa);
        current = current->next;
    }
}

void freeStudents(Student head) {
    Student *temp;
    while(head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    Student head = createStudent(1, "Ali", 18.5);
    head->next = createStudent(2, "Sara", 19.2);
    printStudents(head);
    freeStudents(head);
    return 0;
}
```

```
C

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

typedef struct User {
    int userID;
    char username[50];
    int active;
    int courseCount;
    Course *courses;
} User;

User* createUser(int userID, char username[], int active, int courseCount) {
    User* newUser = (User*)malloc(sizeof(User));
    newUser->userID = userID;
    strcpy(newUser->username, username);
    newUser->active = active;
    newUser->courseCount = courseCount;
    newUser->courses = NULL;
    return newUser;
}

void printUsers(User *head) {
    User *current = head;
    while(current != NULL) {
        printf("User ID: %d | Username: %s | Active: %d | Courses: %d\n",
               current->userID, current->username, current->active, current->courseCount);
        current = current->next;
    }
}

void freeUsers(User head) {
    User *temp;
    while(head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    User users[2] = {
        {1, "Ali", 1, 2}, {2, "Sara", 1, 3}
    };
    printUsers(users);
    freeUsers(users);
    return 0;
}
```

```
C

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

typedef struct A {
    char c;
    int x;
    short s;
} A;

typedef struct B {
    int x;
    short s;
    char c;
} B;

int main() {
    printf("Size of struct A: %zu bytes\n", sizeof(struct A));
    printf("Size of struct B: %zu bytes\n", sizeof(struct B));
    return 0;
}
```

```
C

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

typedef struct Course {
    int courseID;
    char courseName[50];
    int instructorID;
    int sessions;
} Course;

Course* createCourse(int courseID, char courseName[], int instructorID, int sessions) {
    Course* newCourse = (Course*)malloc(sizeof(Course));
    newCourse->courseID = courseID;
    strcpy(newCourse->courseName, courseName);
    newCourse->instructorID = instructorID;
    newCourse->sessions = sessions;
    return newCourse;
}

int main() {
    Course courses[3] = {
        {1, "C Programming", 1, 2}, {2, "Data Structures", 2, 3}, {3, "Operating Systems", 3, 4}
    };
    for(int i = 0; i < coursesCount; i++) {
        printf("Course ID: %d | Name: %s | Instructor ID: %d | Sessions: %d\n",
               courses[i].courseID, courses[i].courseName, courses[i].instructorID, courses[i].sessions);
    }
    freeCourses(courses);
    return 0;
}
```

نمایش سوال



نمایش سوال



نمایش سوال



نمایش سوال



نمایش سوال



و در آخر ...

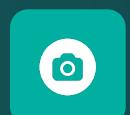
اگر آرایه ها بدن C هستند ،

. مغز آن هستند . Structure

Resources



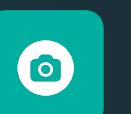
greeksgreeks.com



[Online GBD](#)



w3schools.com



[ChatGPT](#)

با تشکر از توجه شما

End of Slide 