نیمسال اول	عنوان:	استاد : دکتر مرادی
99-98	لیست‌های پیوندی	

لیست های مرتبط[1]:

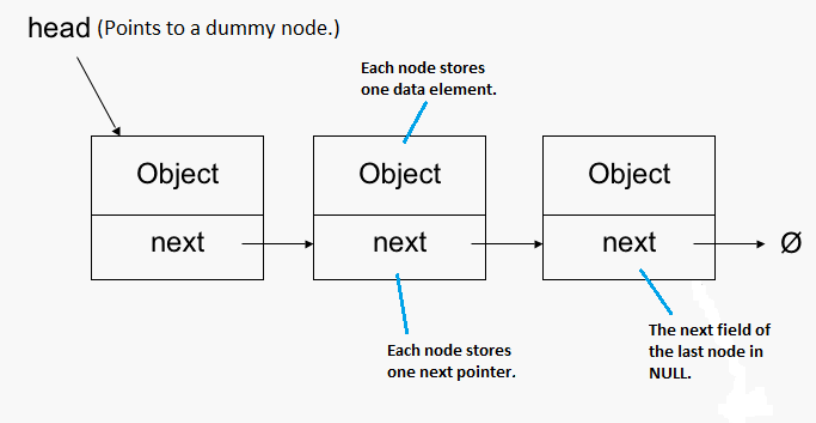
با مفهوم struct در آزمایش‌های قبلی آشنا شدید. از ترکیب مفاهیم اشاره گر، تخصیص حافظه پویا و struct ها می توان لیست های مرتبط را ساخت. لیست‌های مرتبط از تعدادی گره[2] تشکیل می‌شوند که هر گره دارای آدرس گره‌ی بعدی است که این مفهوم در برنامه‌نویسی توسط اشاره‌گرها پیاده سازی می‌شود.

لیست‌های مرتبط کاربردهای بسیاری در نوشتن برنامه ها دارند. از مزایای این ساختار می‌توان به موارد زیر اشاره نمود :

1) قابلیت گسترش اندازه‌ی ساختار به صورت پویا
2) قابلیت حذف و اضافه کردن گره‌ها در وسط لیست
3) قابلیت پیاده‌سازی ساختار هایی مانند صف و پشته با لیست مرتبط
4) عدم نیاز به تعریف اندازه‌ی اولیه

البته این ساختار معایبی نیز دارد. به نظر شما این معایب چیست؟

به شکل زیر دقت کنید تا با ساختار یک لیست مرتبط آشنا شوید:



معمولاً در پیاده سازی لیست های مرتبط (همانند شکل بالا) برای سادگی یک گره[3]ی بیهوده[4] ایجاد می کنند و اشاره گری که به آن اشاره می کند را سر[5] لیست مرتبط گویند. در قسمت Object این گره ی بیهوده هیچ اطلاعات معتبری وجود ندارد و در واقع

---

[1] Linked List
[2] Node
[3] node
[4] dummy
[5] head

لیست ما از عنصر بعد از این گره (عنصر دوم در شکل بالا) شروع می شود. در انجام دهید بعد به علت استفاده از این گره ی تهی پی خواهید برد.

0. انجام دهید (اختیاری)

فایلی به نام WarmUp.c در کنار این فایل قرار دارد که به بررسی دقیق ۴ مساله ساده اما مهم از بحث لیست های مرتبط میپردازد. توصیه میشود قبل از شروع آزمایش آن ها را بررسی کنید.

1. انجام دهید

می خواهیم از ساختاری که برای معرفی یک دانشجوی مبانی تعریف کرده‌ایم، برای ساخت یک لیست مرتبط استفاده کنیم. این ساختمان داده را در قطعه کد زیر مشاهده می‌کنید:

```
typedef struct icsp_student icsp_std;

struct icsp_student {
	char* first_name;
	char* last_name;
	char* student_number;
	float mid_term_exam_score;
	float final_exam_score;
	float homework_score;

};
```

چیزی که در این سوال بر عهده شما گذاشته شده عبارتست از:

تغییر دادن این ساختمان داده به شکلی که بتوان عنصر یک لیست پیوندی را ایجاد کرد. ضمن اینکه تنها قابلیت دسترسی به عنصر بعدی را اضافه کنید. نیازی به داشتن دسترسی به عنصر قبلی نیست.

قسمت۱: مورد خواسته شده را انجام داده و نتایج به دست آمده را در کادر زیر بنویسید..

```
typedef struct icsp_student icsp_std;

struct icsp_student {
	char* first_name;
	char* last_name;
	char* student_number;
	float mid_term_exam_score;
	float final_exam_score;
	float homework_score;
	icsp_std* next_student;
};
```

با اضافه کردن آدرس یک دانشجو می توان آدرس دانشجوی بعدی را ذخیره کرد و در نتیجه می توان در لیست پیوندی از این عنصر استفاده کرد.

2. انجام دهید

برای کار با لیست‌های مرتبط عملیات‌های مورد نظر را به صورت توابع پیاده‌سازی می‌کنیم. همانطور که مطرح شد برای ساخت یک لیست مرتبط ابتدا باید اولین گره‌ی آن را به صورت یک گره‌ی بیهوده تعریف نمود. به همین منظور :

1) تابعی تعریف کنید که ورودی نداشته و مقدار بازگشتی آن از نوع دانشجو تعریف شده باشد.
2) در این تابع یک دانشجو به صورت پویا ایجاد کنید که اطلاعات داخل آن مقدار خاصی نداشته باشند.
3) مقدار اشاره‌گر دانشجوی "بعدی" را برابر NULL قرار دهید.
4) اشاره‌گر دانشجوی ایجاد شده را برگردانید.

نکته: برای تعیین انتهای لیست مرتبط مقدار اشاره‌گر دانشجوی بعدی را برابر NULL قرار می‌دهیم. پس اگر در پیمایش لیست مرتبط بخواهیم رسیدن به انتهای لیست را بررسی کنیم،NULL بودن اشاره‌گرد بعدی را بررسی می‌کنیم. به همین منظور:

1) تابعی بنویسید که اشاره‌گری از نوع دانشجوی دریافت کند و مقدار بازگشتی آن int باشد.
2) در صورتی که دانشجو ورودی آخرین دانشجوی در لیست بود مقدار 1 و در غیر اینصورت مقدار 0 برگردانده شود.

قسمت۲: نتیجه و موارد خواسته شده را در کادر زیر نشان دهید.

```
icsp_std* create_linked_list(){
	icsp_std* head = malloc(sizeof(icsp_std));
	head->next_student = NULL;
	return head;
}
```

با استفاده از تابع بالا می توان اولین گره لیست پیوندی را ایجاد کرد.

```
int is_last(icsp_std* student){
	if (student->next_student == NULL){
		return 1;
	}
	return 0;
}
```

با استفاده از این تابع می توان فهمید که دانشجو، آخرین عضو لیست پیوندی هست یا خیر که از برابری دانشجوی بعد از آن در لیست با NULL فهمیده می شود.

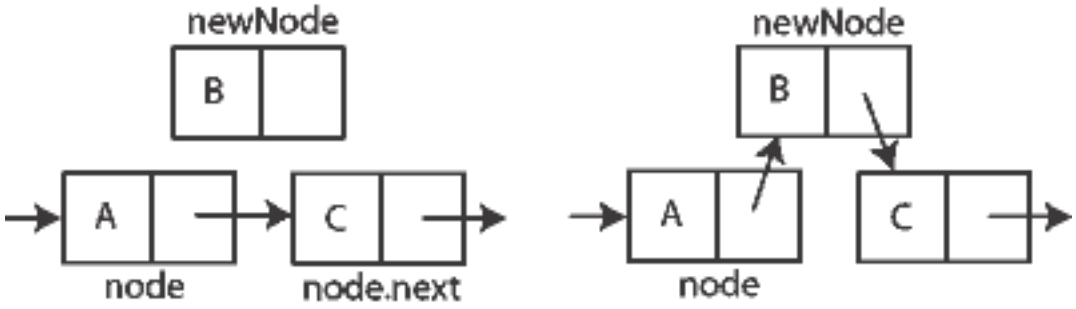
3. انجام دهید

اکنون می‌خواهیم یک لیست ایجاد کرده و به ترتیب، دانشجوهایی را به ابتدای آن اضافه کنیم. برای این کار، ابتدا باید تابعی تعریف کنیم که به وسیله‌ی آن بتوانیم به لیست‌مان گره (در اینجا دانشجو) اضافه کنیم. به قطعه کد زیر که به همین منظور نوشته شده توجه کنید. چه ایرادی در آن مشاهده می‌کنید؟ آن را برای دستیاران آموزشی توضیح دهید.

```
int set_new_std_next_of_head(icsp_std* head_of_list, icsp_std* new_std) {
	if (new_std == NULL || head_of_list == NULL)
		return 0;
	head_of_list->next = new_std;
	new_std->next = head_of_list->next;
	return 1;

}
```

حال می‌خواهیم تابع دیگری تعریف کنیم تا گره‌ای را در وسط لیست اضافه کند. فرآیند انجام این کار را در عکس زیر مشاهده می‌کنید.



با کمک گرفتن از شکل بالا، قطعه کد زیر را تکمیل کنید.

```
int add_to_i(icsp_std* head_of_list, icsp_std* new_std, int i) {
	if (head_of_list == NULL)
		return ZERO;
	icsp_std* current_std = head_of_list->next;
	if (current_std == NULL && i == ZERO) {
		// Write down your code right here and make it short.

		return ONE;
	}else if(current_std == NULL && i != ZERO){
		return ZERO;
	}
	int counter = ZERO;
	while (true){
		if (counter == i) {
			// Write down your code right here
	return ONE;
		}
		counter++;
		if (current_std->next == NULL && counter == i) {
			// Write down your code right here
			return ONE;
		}
		if (current_std->next == NULL && counter < i) {
			return ZERO;
		}
		current_std = current_std->next;
```

```
        }
}
```

قسمت۳: نتیجه و موارد خواسته شده را در زیر نشان دهید.

مشکل کد اول در اینجاست که لیست عضو آخری ندارد یعنی در واقع در کد head به عضو جدید اشاره می کند ولی سپس عضو جدید به خودش اشاره می کند و در نتیجه با این کار نمی توان به اعضای بعدی دسترسی داشت و هر بار که عضو بعدی عضو جدید را صدا می زنیم دوباره به خود عضو جدید می رسیم و شکل آپدیت کردن لیست اشتباه است و در واقع جای دو خط باید عوض شود و به شکل زیر در آید.

```
int set_new_std_next_of_head(icsp_std* head_of_list, icsp_std* new_std) {
	if (new_std == NULL || head_of_list == NULL)
		return 0;
	new_std->next = head_of_list->next;
	head_of_list->next = new_std;
	return 1;
}
```

در کد زیر عضو جدید در جایگاه i قرار می گیرد و به لیست اضافه می شود.

```
int add_to_i(icsp_std* head_of_list, icsp_std* new_std, int i) {
	if (head_of_list == NULL)
		return ZERO;
	icsp_std* current_std = head_of_list->next;
	if (current_std == NULL && i == ZERO) {
		head_of_list->next = new_std;
		new_std->next = NULL;
		return ONE;
	}
	else if (current_std == NULL && i != ZERO){
		return ZERO;
	}
	int counter = ZERO;
	while (1){
		if (counter == i) {
			new_std->next = current_std->next;
			current_std->next = new_std;
			return ONE;
		}
		counter++;
		if (current_std->next == NULL && counter == i) {
			current_std->next = new_std;
			new_std->next = NULL;
			return ONE;
		}
		if (current_std->next == NULL && counter < i) {
			return ZERO;
		}
		current_std = current_std->next;
	}
}
```

4. انجام دهید

حال می‌خواهیم از توابع ایجاد شده استفاده کنیم. آموزش دانشکده لیستی از دانشجویان را در قالب یک فایل (input.txt) تحویل داده است تا وارد کامپیوتر شود. می خواهیم لیست مورد نظر را با استفاده از لیست مرتبط پیاده‌سازی کنیم. برای انجام این کار موارد زیر را پیاده‌سازی کنید:

1) ابتدا با استفاده از تابع تعریف شده یک لیست جدید بسازید.
2) فایل مورد نظر را باز کنید. توجه داشته باشید که می‌توانید برای استفاده از فایل، مسیر کامل را به برنامه‌تان بدهید تا راحت‌تر کار کنید.
3) با استفاده از توابعی که تعریف کردید دانشجویان لیست را به ترتیب از فایل خوانده و به ابتدای لیست اضافه کنید. لازم به ذکر است که برخی از توابع مورد نیاز شما در اختیارتان قرار گرفته است.
4) یک دانشجوی جدید با اطلاعات خودتان ایجاد کنید و به وسط لیست اضافه کنید. روند اجرای کد در قسمت اضافه شدن گره جدید را در debug mode ببینید تا با نحوه‌ی پیمایش روی یک لیست پیوندی و همچنین ساختار یک struct بیشتر آشنا شوید.

قسمت4: نتیجه و موارد خواسته شده را در زیر نشان دهید

```
icsp_std* read_students_credentials_from_file(char* file_name) {
	FILE* input;
	icsp_std* list_students = create_linked_list();
	if (input = fopen(file_name, "r")) {
		int num_of_lines = atoi(read_line_from_input_file(input));
		for (int i = 0; i < num_of_lines; i++) {
			icsp_std* std = (icsp_std*)malloc(sizeof(icsp_std));
			std->first_name = read_line_from_input_file(input);
			std->last_name = read_line_from_input_file(input);
			std->student_number = read_line_from_input_file(input);
			std->mid_term_exam_score = atof(read_line_from_input_file(input));
			std->final_exam_score = atof(read_line_from_input_file(input));
			std->homework_score = atof(read_line_from_input_file(input));
			set_new_std_next_of_head(list_students, std);
		}
		fclose(input);
		return list_students;
	}
	return NULL;
}
```

با اندکی تغییر در کد تابع داده شده بالا می توان لیست پیوندی ایجاد کرده دانشجوی جدید را هر بار بعد از خواندن از فایل به ابتدای لیست پیوندی اضافه کرده و در نهایت بعد از اضافه کردن تمام دانشجویان به لیست پیوندی تابع این لیست پیوندی را برمی گرداند. کافیست در main این تابع را با مسیری که فایل ورودی در آن قرار دارد صدابزنیم و لیست پیوندی را به دست بیاوریم.

برای اضافه کردن دانشجویی در وسط لیست میز می توان از تابع add_to_i استفاده کرده و با قرار دادن 1 برای مقدار i با توجه به این که 3 نفر در لیست هستند دانشجوی جدید را در وسط لیست اضافه کنیم.

5. انجام دهید

با توجه به این که روزانه درخواست‌های متعددی به آموزش دانشکده می‌شود، آموزش می‌خواهد تا با داشتن شماره‌ی دانشجویی دانشجویان بقیه‌ی اطلاعات آن‌ها را پیدا کند. به همین منظور می‌خواهیم تابعی بنویسیم تا با دریافت شماره‌ی دانشجویی اطلاعات دانشجوی مورد نظر را چاپ کند. برای انجام این کار :

1) تابعی بنویسید که اشاره‌گر از نوع دانشجو به عنوان سر لیست و یک شماره‌ی دانشجویی دریافت می‌کند و خروجی آن از نوع اشاره‌گر به دانشجو است.
2) در این تابع لیست مرتبط را پیمایش کنید و در صورتی که شماره‌ی دانشجویی گره‌ی فعلی با ورودی برابر بود همان گره را به عنوان خروجی برگرداند.
3) در صورتی که دانشجوی مورد نظر در لیست وجود نداشت با چاپ پیام مناسب مقدار NULL را برگرداند.
4) تابع مورد نظر را در تابع main آزمایش کنید.

قسمت5 نتیجه و موارد خواسته شده را در زیر نشان دهید

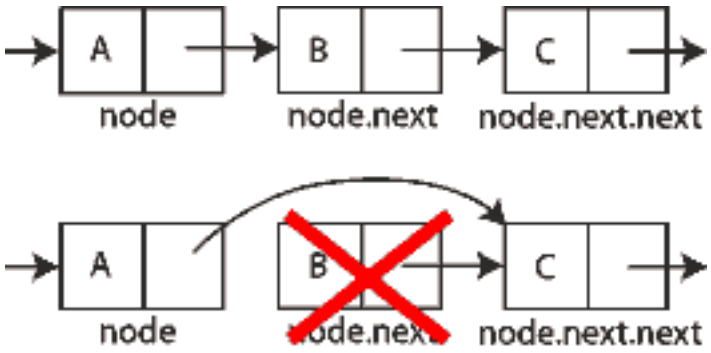
```
icsp_std* find_student(icsp_std* list, int student_id){
	icsp_std* current_std = list->next;
	while (current_std != NULL){
		if (atoi(current_std->student_number) == student_id){
			return current_std;
		}
		current_std = current_std->next;
	}
	printf("No Student Found!\n");
	return NULL;
}
```

با طی کردن لیست پیوندی در صورتی که دانشجو پیدا شد اطلاعات دانشجو برگردانده می شود و در صورت پیدا نشدن با نمایش پیامی مقدار NULL برگردانده می شود.

6. انجام دهید

با توجه به این موضوع که تعدادی از دانشجویان درس را حذف کرده‌اند، می‌خواهیم تعدادی از گره‌های لیست مرتبط ساخته شده را حذف کنیم. عملیات حذف کردن از میانه‌ی لیست دقیقا معکوس عمل اضافه کردن گره در میانه‌ی لیست است.

1) تابعی بنویسید که اشاره‌گر از نوع دانشجو به عنوان سر لیست و یک شماره‌ی دانشجویی دریافت می‌کند و خروجی آن از نوع int است.
2) در این تابع لیست مرتبط را پیمایش کنید و در صورتی که شماره‌ی دانشجویی گره‌ی فعلی با ورودی برابر بود ابتدا آدرس دانشجوی بعدی گره‌ی قبلی را برابر با آدرس دانشجوی بعدی گره‌ی فعلی قرار دهید. سپس اشاره‌گر دانشجوی فعلی را آزاد کنید.



3) در صورت موفقیت عدد 1 و در غیر اینصورت عدد 0 را برگردانید.
4) تابع مورد نظر را در تابع main آزمایش کنید.

قسمت6: نتیجه و موارد خواسته شده را در زیر نشان دهید.

```
int delete_student(icsp_std* list, int student_id){
	icsp_std* current_std = list->next;
	icsp_std* prev_std = list;
	while (current_std != NULL){
		if (atoi(current_std->student_number) == student_id){
			prev_std->next = current_std->next;
			free(current_std);
			return 1;
		}
		current_std = current_std->next;
		prev_std = prev_std->next;
	}
	return 0;
}
```

با طی کردن لیست در صورت پیدا کردن عضو آن را حذف کرده و مقدار یک برگردانده می شود و در صورت پیدا نشدن مقدار صفر برگردانده می شود.

7. انجام دهید (امتیازی)

هر یک از توابع زیر را پیاده‌سازی کنید.

```
int print_reverse(icsp_std *head);
```

اطلاعات دانشجویان لیست را از آخر به اول چاپ کند و در صورت موفقیت عدد 1 و در غیر این‌صورت (برای مثال خالی بودن لیست) عدد 0 را برگرداند. تابع فوق باید به صورت بازگشتی پیاده‌سازی شود.

```
int sort_by_id(icsp_std *head);
```

عناصر لیست را بر اساس شماره‌ی دانشجویی مرتب کنید. مقادیر بازگشتی مانند قسمت‌های قبل است.

```
int sort_by_name(icsp_std *head);
```

عناصر لیست را بر اساس نام خانوادگی مرتب کنید. مقادیر بازگشتی مانند قسمت‌های قبل است.

```
int delete_all(icsp_std *head);
```

تمامی عناصر لیست را حذف کنید. مقادیر بازگشتی مانند قسمت‌های قبل است. تابع فوق باید به صورت بازگشتی پیاده‌سازی شود.

```
int list_length(icsp_std *head);
```

طول لیست مورد نظر را برگرداند.(تعداد عناصر داخل لیست)

قسمت7: نتیجه و موارد خواسته شده را در زیر نشان دهید

```c
int list_length(icsp_std *head){
	icsp_std* current = head->next;
	int count = 0;
	while (current != NULL){
		count++;
		current = current->next;
	}
	return count;
}
```

```c
int print_reverse(icsp_std *head){
	if (head == NULL){
		return 0;
	}
	print_reverse(head->next);
	printf("%s\n", head->first_name);
	printf("%s\n", head->last_name);
	printf("%s\n", head->student_number);
	printf("%.2f\n", head->mid_term_exam_score);
	printf("%.2f\n", head->final_exam_score);
	printf("%.2f\n", head->homework_score);
	printf("====================\n");
	return 1;
}

int main(){
	icsp_std* list = read_students_credentials_from_file(textPath);
	print_reverse(list->next);
	return 0;
}
```

```c
int delete_all(icsp_std *head){
	if (head == NULL){
		return 0;
	}
	delete_all(head->next);
	free(head);
	return 1;
}

int main(){
	icsp_std* list = read_students_credentials_from_file(textPath);
	delete_all(list->next);
	return 0;
}
```