به نام خدا

دانشکده‌ی مهندسی برق و کامپیوتر دانشکده فنی دانشگاه تهران

مبانی کامپیوتر و برنامه‌نویسی

نیمسال اول	عنوان:	استاد : دکتر مرادی
۹۸-۹۹	لیست‌های پیوندی	

لیست های مرتبط[1]:

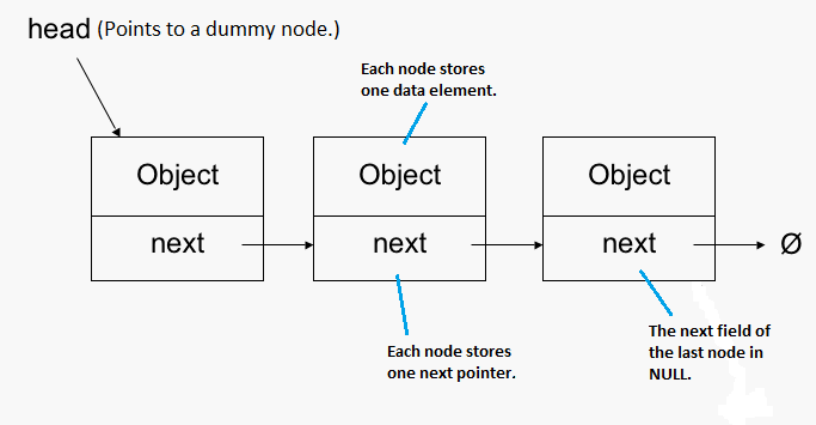
با مفهوم struct در آزمایش‌های قبلی آشنا شدید. از ترکیب مفاهیم اشاره گر، تخصیص حافظه پویا و struct ها می توان لیست های مرتبط را ساخت. لیست‌های مرتبط از تعدادی گره[2] تشکیل می‌شوند که هر گره دارای آدرس گره‌ی بعدی است که این مفهوم در برنامه‌نویسی توسط اشاره‌گرها پیاده سازی می‌شود.

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

۱) قابلیت گسترش اندازه‌ی ساختار به صورت پویا
۲) قابلیت حذف و اضافه کردن گره‌ها در وسط لیست
۳) قابلیت پیاده‌سازی ساختار هایی مانند صف و پشته با لیست مرتبط
۴) 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

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.

۰. 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.

۱. 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..

با اضافه کردن یک استراکچر پوینت از نوع دانشجو داخل تعریف استراکچر دانشجو میتوان ساختار را به شکلی تغییر داد تا بتوان عنصر یک لیست پیوندی را ایجاد کرد.

```
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;
};
```

۲. 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 :

۱) تابعی تعریف کنید که ورودی نداشته و مقدار بازگشتی آن از نوع دانشجو تعریف شده باشد.
۲) در این تابع یک دانشجو به صورت پویا ایجاد کنید که اطلاعات داخل آن مقدار خاصی نداشته باشند.
۳) مقدار اشاره‌گر دانشجوی "بعدی" را برابر NULL قرار دهید.
۴) 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:

۱) تابعی بنویسید که اشاره‌گری از نوع دانشجوی دریافت کند و مقدار بازگشتی آن int باشد.
۲) در صورتی که دانشجو ورودی آخرین دانشجوی در لیست بود مقدار ۱ و در غیر اینصورت مقدار ۰ برگردانده شود.

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

ایجاد گره اولیه و بیهوده به شکل زیر خواهد بود:

```
icsp_std *dummy_std() {
	icsp_std *new_std = (icsp_std *)malloc(sizeof(icsp_std));
	new_std->next = NULL;
	return new_std;
}
```

چک کردن اینکه آیا دانشجوی ورودی آخرین دانشجو هست یا خیر:

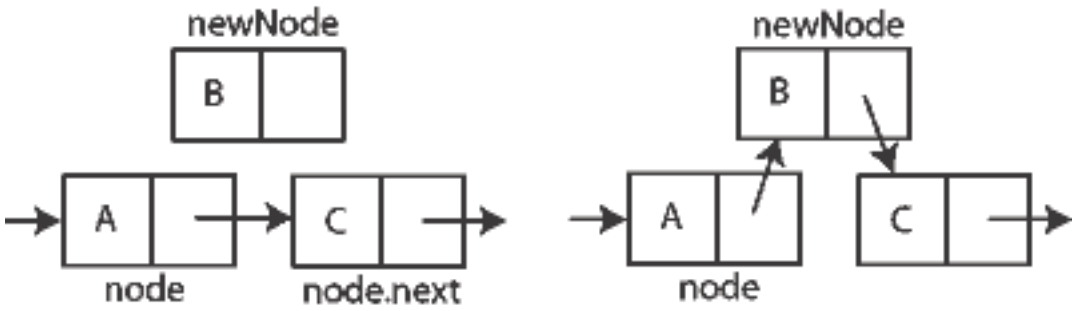
```
int end_check(icsp_std* std) {
	if (std->next == NULL)
		return 1;
	return 0;
}
```

۳. 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.

ایراد قطعه کد اول این است که تمام گره هایی را که میخواهیم اضافه کنیم در `head_of_list->next` قرار میدهد که این یعنی اگر دانشجویی را اضافه کنیم آنرا در گره بعد `head_of_list` قرار میدهد و دانشجوی بعدی را نیز باز در همان جا قرار میدهد و بعدی و بعدی و بعدی که این یعنی از دست رفتن دانشجوهای قبلی. همچنین این قطعه کد حتی برای اضافه کردن یک دانشجو هم ایراد دارد زیرا باید گره بعدی دانشجوی اضافه شده را نول قرار داد ولی این اتفاق نیفتاده است.

قطعه کد اصلاح شده برای قسمت دوم:

```
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 (true) {
            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;
      }
}
```

۴. 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:

۱) ابتدا با استفاده از تابع تعریف شده یک لیست جدید بسازید.
۲) 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.
۳) 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.
۴) 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.

قسمت۴: نتیجه و موارد خواسته شده را در زیر نشان دهید

قسمتی از تابع `read_students_credentials_from_file` که مشکل داشت به شکل زیر قابل اصلاح است:

```
// Somthing is missing. find it! <<-----------------------------------<<
students[i] = std;
if (i != 0)
{
	students[i]->next = NULL;
	students[i - 1]->next = students[i];
}
```

تابع اصلی به شکل زیر خواهد بود:

```
int main() {
	char* file_name = (char*)malloc(9 * sizeof(char));
	strcpy(file_name, "input.txt");
	icsp_std** M = read_students_credentials_from_file(file_name);
	icsp_std* new_std = (icsp_std*)malloc(sizeof(icsp_std));
	new_std->first_name = (char*)malloc(5 * sizeof(char));
	new_std->last_name = (char*)malloc(3 * sizeof(char));
	new_std->student_number = (char*)malloc(9 * sizeof(char));
	strcpy(new_std->first_name, "mahdi");
	strcpy(new_std->last_name, "MWM");
	strcpy(new_std->student_number, "810198434");
	new_std->mid_term_exam_score = 17;
	new_std->final_exam_score = 19;
	new_std->homework_score = 19;
	add_to_i(M[0], new_std, 0);
}
```

البته کد با فرض اینکه گره اول بیهوده است نوشته شده و در واقع با اجرای کد بالا دانشجوی جدید بعد دانشجوی دوم قرار میگرد که اگر گره اول بیهوده بود بدرستی بعد از دانشجوی اول قرار میگرفت.

۵. 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 :

۱) 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.
۲) 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.
۳) در صورتی که دانشجوی مورد نظر در لیست وجود نداشت با چاپ پیام مناسب مقدار NULL را برگرداند.
۴) تابع مورد نظر را در تابع main آزمایش کنید.

قسمت۵ نتیجه و موارد خواسته شده را در زیر نشان دهید

کد خواسته شده بصورت زیر خواهد بود:

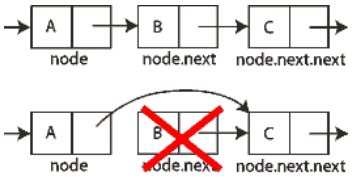
```
icsp_std* std_num_detector(icsp_std* head, char* std_num) {
	while (head != NULL)
	{
		if (!strcmp(head->student_number,std_num))
			return head;
		head = head->next;
	}
	printf("There is no student with that student-number.\n");
	return NULL;
}

void print_info(icsp_std* std) {
	printf("First Name: %s\n", std->first_name);
	printf("Last Name: %s\n", std->last_name);
	printf("Student Number: %s\n", std->student_number);
	printf("Mid Term Exam Score: %f\n", std->mid_term_exam_score);
	printf("Final Exam Score: %f\n", std->final_exam_score);
	printf("Homework Score: %f\n", std->homework_score);
}

int main() {
	char* file_name = (char*)malloc(9 * sizeof(char));
	strcpy(file_name, "input.txt");
	icsp_std** M = read_students_credentials_from_file(file_name);
	printf("Enter Student Number: ");
	char std_num1[9];
	scanf("%s", std_num1);
	icsp_std* NEW = std_num_detector(M[0], std_num1);
	print_info(NEW);
}
```

۶. 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.

۱) تابعی بنویسید که اشاره‌گر از نوع دانشجو به عنوان سر لیست و یک شماره‌ی دانشجویی دریافت می‌کند و خروجی آن از نوع int است.

۲) 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.



۳) در صورت موفقیت عدد ۱ و در غیر اینصورت عدد ۰ را برگردانید.

۴) تابع مورد نظر را در تابع main آزمایش کنید. **قسمت۶**: نتیجه و موارد خواسته شده را در زیر نشان دهید.

کد خواسته شده بصورت زیر خواهد بود:

```
int delete_student(icsp_std* head, char* std_num) {
	icsp_std* Before_node = head;
	while (head != NULL)
	{
		if (!strcmp(head->student_number, std_num))
		{
			if (head->next != NULL)
			{
				Before_node->next = head->next;
				free(head);
				return 1;
			}
		}
		Before_node = head;
		head = head->next;
	}
	return 0;
}
int main() {
	char* file_name = (char*)malloc(9 * sizeof(char));
	strcpy(file_name, "input.txt");
	icsp_std** M = read_students_credentials_from_file(file_name);
	printf("Enter Student Number: ");
	char std_num1[9];
	scanf("%s", std_num1);
	delete_student(M[0], std_num1);

}
```

۷. 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 ۱ و در غیر این‌صورت (برای مثال خالی بودن لیست) عدد ۰ 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)

قسمت۷: نتیجه و موارد خواسته شده را در زیر نشان دهید.

```
int print_reverse(icsp_std *head) {
	icsp_std* current = head;
	for (int i = list_length(head); i > 0; i--)
	{
		current = head;
		for (int j = 1; j < i; j++)
		{
			current = current->next;
		}
		print_info(current);
	}
	return 1;
}
```

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



int delete_all(icsp_std *head) {
	icsp_std* current = head;
	for (int i = list_length(head); i > 0; i--)
	{
		current = head;
		for (int j = 2; j < i; j++)
		{
			current = current->next;
		}
		free(current->next);
		current->next = NULL;
	}
	return 1;
}
```