کاراکتر:

مفهوم کاراکتر، خارج از زبان سی، به معنای یک نماد (مثلاً a یا 2 یا *) است که بر روی صفحه دیده می‌شود. هر کدام از این نمادها، بنابر استاندارد، دارای شماره‌ی معینی هستند، که به آن «کد اسکی»[1] می‌گویند.

در زبان سی، کاراکتر (**char**) یک نوع متغیر یک‌بایتی است که می‌تواند از 128- تا 127 مقدار بگیرد. اما زمانی که بخواهید یک کاراکتر را چاپ کنید، به جای اینکه مقدار عددی آن چاپ شود، نماد معادل چاپ خواهد شد.

کاراکترهای کنترلی:

در جدول کد های اسکی، ۳۲ کاراکتر ابتدایی، کاراکترهایی هستند که خودشان دارای نماد قابل چاپ نیستند، بلکه معمولاً برای تنظیم ادامه‌ی جریان خروجی بکار می‌روند. به این کاراکترها، کاراکترهای کنترلی می‌گویند. برای مثال، چاپ کاراکتر با کد اسکی 10 باعث می‌شود تا ادامه‌ی خروجی از ابتدای خط جدید شروع شود.

از آنجا که کاراکترهای کنترل دارای نماد نیستند، برای استفاده از آن‌ها در برنامه از ترکیب دو نماد استفاده می‌کنیم. برای مثال، همان‌طور که `cout << 'a';` حرف a را در خروجی می‌نویسد، `cout << '\n';` باعث شروع خط جدید می‌شود. دقت کنید که اگر چه '\n' با دو نماد نمایش داده می‌شود، اما دقیقاً نشان‌دهنده‌ی یک کاراکتر (با مقدار عددی 10) است.

c-string:

یک آرایه از کاراکتر است. تقریباً تمام توابع در هنگام کار با این نوع از رشته، یک اشاره‌گر به ابتدای آرایه را دریافت می‌کنند و اولین جایی که به کاراکتری با کد اسکی 0 برخورد کنند، آنجا را انتهای رشته در نظر می‌گیرند. (کاراکتر با کد 0 معادل '\0' است و با '0' فرق می‌کند. از آنجا که '\0' اعلام‌کننده‌ی پایان رشته است، به آن Terminating Null هم می‌گویند)

سه دستور زیر دقیقاً یکی هستند.

```
char s[] = "test";
char s[] = {'t', 'e', 's', 't', '\0'};
char s[] = {116, 101, 115, 116, 0};
```

دقت کنید که بعد از اجرای هر کدام از دستورهای بالا، s آرایه‌ای از کاراکتر به طول ۵ خواهد بود. (یادآوری: اگر در هنگام تعریف آرایه، اندازه‌ی اولین بعد را تعیین نکنیم، طول این بعد بر اساس مقدار اولیّه‌ای که داده شده تعیین می‌شود)

استفاده از آرایه‌ی کاراکتر به عنوان رشته، معمولاً کمی گیج‌کننده است. اما از آنجا که برخی توابع پرکابرد قدیمی (مانند printf) تنها با این نوع رشته کار می‌کنند، آشنایی با این روش مفید خواهد بود.

[1] ASCII Code

:string

پیاده‌سازی رشته در STL، شباهت زیادی به یک vector<char> دارد، اما توابع بسیاری هستند که باعث می‌شوند کار با string بسیار ساده‌تر از کار با vector<char> باشد. برای تعریف یک رشته، بعد از اضافه کردن دستور محبوب #include <string> به ابتدای کد، می‌نویسیم:

```
string s;
```

بعد از اجرای این دستور، s یک رشته‌ی خالی خواهد بود. از اینجا به بعد، منظورمان از رشته، string خواهد بود. (و نه آرایه‌ی کاراکترها)

مقداردهی به رشته‌ها با عملگر = و اضافه کردن یک رشته به انتهای رشته‌ی دیگر با عملگر + انجام می‌شود. دسترسی به iمین کاراکتر یک رشته مثل آرایه (یا vector) و با کمک عملگر [ ] انجام می‌شود. همچنین چاپ کردن رشته‌ها توسط جریان‌ها (مثل cout و cerr) به سادگی انجام می‌شود.

کاربرد ساده‌ای از این اعمال در نمونه‌کد زیر آورده شده:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	string s = "salam", p;
	p = "hel";
	string t = s + "=" + p + p[2] + "o";
	t += "!";
	cout << t << endl;
	return 0;
}
```

برخی توابع پرکاربرد:

- s.size() و s.length()

  مقدار بازگشتی این توابع، طول رشته‌ی s خواهد بود.

- s.substr(p, n)

  مقدار بازگشتی این تابع، زیررشته‌ای از s و به طول n خواهد بود که از خانه‌ی p آغاز می‌شود.

- s.find(p)

  این تابع، رشته‌ی p را در رشته‌ی s جستجو می‌کند. در صورتی که آن را بیابد، شماره‌ی کوچکترین خانه‌ای از s را بازمی‌گرداند، که آغازکننده‌ی p باشد. در غیر این صورت، مقدار ثابت string::npos را بازمی‌گرداند.

رشته‌خوانی:

برای خواندن رشته از ورودی، می‌توانید از دستور `cin >> s;` استفاده کنید. دقت کنید که `cin` در حالت عادی، خواندن رشته را تا زمان برخورد با کاراکترهایی مانند `space` و `newline` ادامه می‌دهد.

بعضی اوقات، نیاز هست که یک خط از ورودی را عیناً (با حفظ `space`های میانی) ذخیره کنیم. در این موارد، می‌توانیم از تابع `getline` استفاده کنیم. برای مثال، دستور `getline(cin, s);` از محل کنونی ورودی، تا انتهای خط را می‌خواند و در `s` ذخیره می‌کند.

مقایسه‌ی رشته‌ها:

رشته‌ها قابل مقایسه هستند. برای مثال، اگر دو رشته‌ی `s` و `p` داشته باشیم، می‌توانیم با چک کردن شرط `s == p` بفهمیم که دو رشته برابر هستند یا نه. همچنین، بنا به ترتیب پیش‌فرضی که برای رشته‌ها در نظر گرفته شده، می‌توان با نوشتن `s < p` آنها را مقایسه کرد.
ترتیبی که برای مقایسه‌ی رشته‌ها بکار می‌رود، همان ترتیب کلمات در دیکشنری است. برای مثال:

```
"aa" < "ab" < "abc" < "ax" < "b"
```

به این ترتیب‌دهی (که قابلیت تعمیم به هر دنباله‌ای از اعداد را دارد)، ترتیب لغت‌نامه‌ای[2] می‌گویند.

توجه!

مجددا دقت کنید که اکثر مطالبی که گفته شد در مورد `string` بوده، و در بسیاری از موارد در مورد آرایه‌های کاراکتری کاربردی ندارد. برای مثال، اگر دو رشته را به صورت آرایه‌ی کاراکتری ذخیره کرده باشید، نمی‌توانید از `==` یا `<` برای مقایسه‌ی آنها استفاده کنید. همچنین امکان متصل کردن این دو رشته با استفاده از عملگر `+` وجود ندارد. برای انجام این اعمال نیاز به توابع متفاوتی خواهید داشت که اکثرا در کتابخانه‌ی **`cstring`** تعریف شده‌اند.

[2] Lexicographical Order

تمرین:

۱. برنامه‌ای بنویسید که بعد از اجرا، عبارت زیر را در خروجی بنویسد:

```
HEL\LOWO"RLD
```

۲. خروجی برنامه‌ی زیر به ازای ورودی داده شده چیست؟

برنامه

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	int n;
	string t, s, p;
	cin >> n >> t;
	getline(cin, s);
	getline(cin, p);
	cout << n << endl << t << endl << s << endl << p << endl;
	return 0;
}
```

ورودی

```
44 sh44zzz
Shaazzz.blogfa.com
STR
```

۳. برنامه‌ای بنویسید که یک عدد در مبنای ۱۰ دریافت کند و آن را در مبنای ۱۵ چاپ کند. (ارقام ۱۰ تا ۱۴ را با a تا e نشان دهید)

۴. تابعی بنویسید که یک **int** دریافت کند و معادل رشته‌ای (string) آن را بازگرداند.

۵. برنامه‌ای بنویسید که ابتدا n را دریافت کند. سپس n نام (که فقط شامل حروف انگلیسی هستند) را از ورودی بخواند و آنها را بر اساس حروف الفبا مرتب کند.