پيوست آ

راهنمای نصب کامیایلر و اجرای برنامههای C++

در این فصل، به روش نصب کامپایلرهای ++C روی کامپیوترهای شخصی و گوشیهای هوشمند و تبلت میپردازیم. در خصوص هر کدام، روی پلتفرمهای معروف جزئیات بیان شده است ولی نصب روی پلتفرمهای دیگر نیز به طور مشابه و با در نظر گرفتن پلتفرم، قابل انجام خواهد بود.

۱.۱ کامپایلر روی کامپیوترهای شخصی

۱.۱.۱ راهنمای نصب کامپایلر Code::Blocks

اگر کاربر ویندوز هستید، ابتدا به یکی از این دو آدرس بروید و فایل نصب را بارگذاری کنید. http://sourceforge.net/projects/codeblocks/files/Binaries/13.12/ Windows/codeblocks-13.12mingw-setup.exe/download http://prdownload.berlios.de/codeblocks/codeblocks-13.12-setup.

exe

برای نصب مراحل زیر را طی کنید:

۱. فایل دانلود شده را اجرا کنید.



- کلید Next را کلیک کنید.
- روی Agree کلیک کنید.



۳. کلید Next را کلیک کنید.

🕞 CodeBlocks Setup	
	hoose Components Choose which features of CodeBlocks you want to install.
Check the components you wa install. Click Next to continue.	ant to install and uncheck the components you don't want to
Select the type of install:	Full: All plugins, all tools, just everything
Or, select the optional components you wish to install:	Pefault install Contrib Plugins C::B Share Config MinGW Compiler Suite
	Description
Space required: 109.9MB	Position your mouse over a component to see its description.
Nullsoft Install System v2.35	
	< Back Next > Cancel

در این مرحله میتوانید محل نصب را انتخاب کنید و سپس کلید Install را فشار

دهيد.





- ۶. با فشار دادن کلید Finish نصب به پایان می رسد.
 CodeBlocks Setup
 Completing the CodeBlocks Setup
 Wizard
 CodeBlocks has been installed on your computer.
 Cick Finish to dose this wizard.
- به طور پیشفرض میانبر Code::Blocks در Desktop ساخته می شود. در غیر اینصورت می توانید در منوی Start آن را پیدا کنید.

۲.۱.۱ نصب و تنظیم Help برای Code::Blocks

در قدم اول نیاز است فایلهای Help نرمافزار را از اینترنت دانلود کنید.

به عنوان مثال میتوانید با مراجعه به این آدرس این فایلها را بارگذاری کنید: http://onnerby.se/~daniel/chm



- ۲. فایل دریافتی را در پوشه محل نصب نرمافزار ذخیره کنید. به عنوان نمونه این محل میتواند C:\Program Files\Codeblocks باشد.
- ۳. بعد از ذخیره فایل در پوشه Code::Blocks روی فایل راست کلیک کرده و properties را انتخاب کنید. سپس فایل را Unblock کنید.

😤 cpp Properti	es 💌
General Secu	rity Details Previous Versions
2	срр
Type of file:	Compiled HTML Help file (.chm)
Opens with:	😵 Microsoft® HTML Help Change
Location:	D:\Program Files (x86)\CodeBlocks
Size:	461 KB (472,251 bytes)
Size on disk:	464 KB (475,136 bytes)
Created:	Today, ۲۰۱۴ , ۱۶ , نوفمبر ۲۰۱۴ , ۲۰۱۴
Modified:	Today, ۲۰۱۴ , ۱۶ نوفمبر ۱۴, ۱۴
Accessed:	Today, ۲۰۱۴ ,۱۶ ,۱۶ ,نوفمبر ۳ minutes ago
Attributes:	Read-only Hidden Advanced
Security:	This file came from another computer and might be blocked to help protect this computer.
	OK Cancel Apply

۴. در قدم بعد در Code::Blocks از منوی Settings گزینه Environment را انتخاب



Help files				
%ENV% export E= %E	*		Add Rename	
Environment variables			Up Down	
HeaderFixup configuration				
R				
Help files				
thread search	н	TP: \$(keyword) will be replaced by the word under the cursor This line default help file (shortcut: F1) This line represents a full command to be executed Open this file with the embedded help viewer (only for HTML files) Preserve keyword case	····	
	•	Default keyword value:		
		OK Cancel		

۵. در سمت چپ پنجره باز شده Help file را انتخاب کنید.

۶. كليد Add را فشار داده و در قسمت اضافه كردن title نام دلخواهي را تايپ كنيد.

👪 main.cpp [Test] - Code::Blocks 12:11	
File Edit View Search Project Build Debug wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help	
🛿 🕐 🕞 🗐 🍕 🐁 🐘 🖿 🔍 🏨 🛛 🎃 🕨 🏶 😵 🖾 Build taraet: Debua 💦 🔹 😽 😪 😂	🧟 G: 🧐 🗉 🔯 🗐
Environment settings	23
Help files	
Management E	Add
✓ Projec SE	Rename
Worksp Environment variables	Delete
	Up
	Down
Add title	
HeaderFixup	E
Please enter new help file title:	
C++ Helpi	
OK Cancel	
Heip hies	
TIP: \$(keyword) will be replaced by the word under the cursor	
This is the default help file (shortcut: F1)	-
This line represents a full command to be executed	
Open this file with the embedded help viewer (only for HTML files)	×
Thread search Preserve keyword case	
Default keyword value:	
OK Cancel	
Command:	- 🖻 🖹 🗶
D:\New folder\Test\main.cpp WINDOWS-1256 Line 7, Column 16 Insert Read/Write	default 🖸

 ۷. با زدن کلید OK در مرحله بعد برای انتخاب مسیر فایل Help در پنجره نمایش داده شده Yes را انتخاب کنید.





This is the default help file (shortcut: FI) را تیک بزنید و سپس OK را انتخاب



راهنمای نصب کامپایلر و اجرای برنامههای ++C

۱۰ تنظیمات به پایان رسید. برای باز کردن این فایل Help کافی است روی کلمه مورد
 نظر در محیط نرمافزار کلیک کرده و کلید FI را فشار دهید.

البته میتوانید بیش از یک فایل را به همین روش جهت Help نرمافزار تنظیم کنید که در این حالت برای یکی از فایل ها کلید FI را به عنوان کلید میانبر انتخاب کنید. برای دیدن بقیه فایلها میتوانید از منوی Help، این راهنماها را انتخاب کنید.

👫 *main.cpp [Test] - Code::Blocks 8.0	02	×
File Edit View Search Project	Build Debug wxSmith Tools Plugins Settings Help	
1 🖻 🖻 🗿 🔦 🔌 🕹 6) 😰 🕒 💁 🚱 🚱 🐼 Build target: Debi 🕜 About	<i>i.</i>
	√ main(): int	-
	Plugins	
Management X	*main.cop × C++ Help F1	4 Þ
Projects Symbols 4	C++ Guid	
- Workspace	3 void f(int a, int b) C++ Manual	
Test	4 🗏 (CodeBlocks	
- Sourcer	5 int suml;	
Jources	6 suml=a+b;	
main.cpp	B b	
	9 int main()	
	10 🖽 (=
	11 int a=5, b=6, sum, d;	
	12 sum=a+b;	
	13 coutesumeenal;	
	15 For (int implies its) (its)	
	16 🗄	
	17 i++;	
	18 cout< <i;< th=""><th></th></i;<>	
	19)	
	20 d=50;	-
	21 f(sum,d);	
	Logs & others	×
	📝 Code::Blocks 🔍 Search results 💊 Build log 🦿 Build messages 💊 Debugger 🔍 Thread search	4 ⊳
	Checking for existence: D:\New folder\Test\bin\Debug\Test.exe	
	Executing: "D:\Frogram Files (x86)\CodeBlocks/cb_console_runner.exe" "D:\New folder\Test\ Dobug\Test eve" (in D:\New folder\Test))	bin
	Process terminated with status () (M minutes 1 seconds)	
	default Line 20, Column 4 Insert Modified Read/Write default	

۳.۱.۱ شیوه ورود برنامه و اجرای آن

نوشتن برنامه در Code::Blocks (روش اول)

۱. از منوی File گزینه New و سپس File را انتخاب کنید. پنجرهٔ زیر ظاهر می شود.

New from templa	ite	x			
Projects Build targets Files Custom User templates	Category: <ali categories=""></ali>	Go Cancel			
		View as			
TIP: Try right-dicking an item					
 Select a wizard type first on the left Select a specific wizard from the main window (filter by categories if needed) Press Go 					

۲. روی آیکون C/C++ Source کلیک کنید. پنجرهٔ زیر ظاهر میشود.



۳. روی Next کلیک کنید. پنجرهٔ زیر ظاهر میشود.

C/C++ source	×
C/C++ FILE	Please select the language for the file. Please make a selection C C++
	< Back Next > Cancel

۴. روی ++C و سپس روی Next کلیک کنید. پنجرهٔ زیر ظاهر می شود.



۵. در این پنجره، روی ... کلیک کنید. پنجره زیر ظاهر می شود. در این پنجره می توانید به مسیر مورد نظر جهت ذخیره کردن برنامه بروید و سپس در قسمت Filename نام فایل را وارد کنید. نام فایل باید انگلیسی و بدون فاصله باشد و همچنین پسوند آن باید cpp. باشد. دقت کنید که با نگذاشتن cpp. به عنوان پسوند فایل، فایل با پسوند c. ایجاد می شود که در مراحل بعدی اجرای برنامه را با مشکل مواجه می کند.

Select filename	Due	-R-	the Codes & Ch04		L. Samt Ch04	_	× _
	nProg	300	ok V Codes V Cho4	•	Search Ch04		7
Organize 🔻 Ne	v folo	ler				· · · · · · · · · · · · · · · · · · ·	0
☆ Favorites	Â	1	Name		Date modified	Туре	^
🥅 Desktop			🕶 abs		۰V/۱۱/۲۰۱۴ ۰۴:۳۶	C++ source file	
〕 Downloads			CH BMM		۰V/۰۴/۲۰۰۹ ۰۲:۳۱	C++ source file	Ξ
📃 Recent Places	Ξ		CH BMM2		۰۲/۱۲/۲۰۱۴ ۰۸:۵۴	C++ source file	
			🕶 Digits		۰V/۰۴/۲۰۰۹ ۰۱:۳۷	C++ source file	
🧊 Libraries			🕶 Digits2		۰V/۰۴/۲۰۰۹ ۰۱:۰۹	C++ source file	
Documents		9	🕶 dowhile		•T/)T/T•)Ť •V:))	C++ source file	
🎝 Music			🕶 even-odd		۰V/۰۴/۲۰۰۹ ۰۲:۲۷	C++ source file	
Pictures			🕶 fact		•٧/•۴/۲٠•۹)•:•۹	C++ source file	
🚼 Videos			🕶 Fibonacci		۰V/۰۴/۲۰۰۹ ۰۱:۳۹	C++ source file	
			🗪 for		•1/11/11/11 •1:19	C++ source file	-
🤣 Homegroup	-	4				4	
File name:	File name: testm.cpp		Type: C++ source Size: 62 bytes	e file		•	
Save as type: C++ files (*.cpp;*.cxc;*.cc)				1/ 1/1+11 +1(14 PIM		-	
A Hide Folders					Save	Cancel	

۶. پس از انتخاب نام فایل، روی Save کلیک کنید. با این کار پنجره بسته شده و از پنجرهٔ قبلی نیز Finish را کلیک کنید. حال میتوانید برنامه خود را بنویسید. برای اجرای آن به قسمت ۲.۱.۱ بروید.

نوشتن برنامه در Code::Blocks (روش دوم)

۱. از منوی File گزینه New و سپس Project را انتخاب کنید.



۲. در کادر باز شده گزینه Console Application را انتخاب کنید. شکل زیر را ببینید.



۳. در پنجرهٔ باز شده به شکل زیر روی Next کلیک کنید.



۴. در کادر باز شده ++C را انتخاب کنید و سپس Next را فشار دهید.

Console application	
🚮 Console	Please select the language you want to use. Please make a selection C C++
	< Back Next > Cancel

۵. در این مرحله نام و محل ذخیره پروژه را مشخص کنید. در قسمت Project title

نام و در قسمت Folder to create project in محل ذخیره را تعیین کنید و سپس

	Nex را فشار دهید.
Console application	—
🐻 Console	Please select the folder where you want the new project to be created as well as its title.
	Project title:
	Folder to create project in:
	Project filename:
	Resulting filename:
	D: wew folder (lest (lest.cdp
	< Back Next > Cancel

۶. پیش فرشهای این مرحله را تغییر نمیدهیم و در نهایت Finish را فشار میدهیم.

Console application	
🚮 Console	Please select the compiler to use and which configurations you want enabled in your project. Compiler:
	GNU GCC Compiler
	Create "Debug" configuration: Debug
	"Debug" options
	Output dir.: bin\Debug\
	Objects output dir.: obj\Debug\
	✓ Create "Release" configuration: Release "Release" options Output dir.: bin\Release\ Objects output dir.: obj\Release\
	< Back Finish Cancel



نرمافزار محیط برنامهنویسی جدیدی را برای شما باز خواهد کرد.

در سمت چپ محیط این نرمافزار، پنل Management را مشاهده میکنید. اولین گزینه در این پنل Project هست. در این قسمت میتوانید نام پروژه خود را ببینید و در پایین آن پوشه Sources وجود دارد. این پوشه را با کلیک کردن + کنار آن باز



در این پوشه فایل main.cpp وجود دارد. این فایل در زمان ایجاد برنامه به طور

خودکار ساخته میشود.

با کلیک کردن روی این فایل و باز کردن آن، محیط برنامهنویسی برای شما در سمت راست باز میشود. دستورهای پیش فرضی در این محیط وجود دارد.

👫 main.cpp [Test] - Code::Blocks 8.02		—
File Edit View Search Project	Build Debug wxSmith Tools Plugins Settings Help	
🔢 🖻 🗃 🎒 🔦 🔖 🔥 🖉	🛍 💁 💁 🖕 🧊 🐼 🖓 🐼 Buld target: Debug 💿 🔹 🖓 🖑 🐼 🗌	· 1.
	₩	•
		_
Management X	main.cpp ×	4 Þ
Projects Symbols 4 Construction of the symbols of	<pre>1 #include <iostream> 2 using namespace std; 4 int main() 6 □ { cout << "Hello world!" << endl; 8 9 10 </iostream></pre>	
	<	•
	ogs & others	×
	🥂 Code::Blods 🔍 Search results 🛇 Build log 🧳 Build messages 💊 Debugger 🔍 Thread search	
D:\New folder\Test\main.cpp	default Line 1, Column 1 Insert Read/Write default	

بعد از نوشتن برنامه ابتدا باید آن را Build کنید. منظور از Build، ترجمهٔ برنامه به زبان ماشین است. برای اجرای آن، لازم است ابتدا برنامه به زبان ماشین تبدیل شود. روش این دو کار در قسمت بعد آمده است.

- **روش ترجمه و اجرای برنامه** برای این کار میتوانید از سه روش استفاده کنید:
- ۱. از منوی Build اولین گزینه را اجرا کنید.
 ۲. همچنین میتوانید با استفاده از کلیدهای ترکیبی Ctrl+F9 این کار را انجام دهید.
 ۳. گزینه Build را در بالای صفحه کلیک کنید. آیکون این کار یک چرخ دندهٔ زرد رنگ است.



برای اجرای برنامه نیاز است که برنامه run شود که برای این کار نیز میتوان به سه روش عمل کرد:

- ۱. از منوی Build گزینه run را انتخاب کنید.
- ۲. از کلیدهای ترکیبی Ctrl+F10 استفاده کنید.
- ۳. گزینه Run را در بالای صفحه اجرا کنید. آیکون اجرا مثلث سبز رنگ است.

-					
💾 main.cpp [Test] - Code::Blocks 8.02					• ×
File Edit View Search Project	Build Debug	wxSmith Tools Plugins Setti	ngs Help		
10 🖻 🖬 🗿 🔦 🔌 🔏 🖻	🛍 🔍 🖻	i 💊 📐 🥸 🐼 🐼 Build target:	Debug	💶 🖬 🦷 😚 🗈 🏷 🕼 😣	🗖 i.
		Rup			•
		(Min)	1	- <u>Q</u>	
Management X	main.cpp ×				4 Þ
Projects Symbols 4 → ● ● Workspace ● ● Sources ● ● Sources ● ● Sources	1 1 1 1 2 3 4 5 6 7 9 10 10 10 10 10 10 10 10 10 10	<pre>include <lostream> using namespace std; int main() { cout << "Hello world!" } m dds Q Search results D Build k </lostream></pre>	< endl; g 🔮 Buld messages	Debugger Q Thread search	• • • • • • • • • • • • • • • • • • •
Run the active project	default	Line 1, Column 1	Insert	Read/Write default	

البته میتوانید اعمال Build و Run را با هم اجرا کنید. که برای این کار نیز میتوان از سه طریق عمل کرد:

- از منوی Build and run را انتخاب کنید.
 - ۲. از کلید F9 استفاده کنید.
- ۳. گزینه Build and run را در بالای صفحه اجرا کنید.



با اجرای برنامه خروجی در صفحهای مشکی برای شما نمایش داده خواهد شد.



۴.۱.۱ خطاگیری- گرامری و منطقی

خطاهای برنامه را میتوان به دو دسته تقسیم کرد:

- . خطای گرامری
- ۲. خطای منطقی

خطای گرامری

این دسته از خطاها به خطاهایی مربوط میشود که معمولاً کاربران مبتدی در نوشتن دستورات مرتکب میشوند و دستورات را به شکل صحیح نمینویسند. بیشترین خطا در این زمینه فراموش کردن نوشتن ؛ (سمی کالن) است. میتوانید نوع خطا و محل خطا را در پایین صفحه در قسمت Build messages مشاهده کنید. در محیط نرمافزار نیز یک مربع قرمز ابتدای خطی که در آن خطا وجود دارد، ظاهر میشود.

💾 main.cpp [Test] - Code::Blocks 8.07		
File Edit View Search Project	Build Debug wxSmith Tools Plugins Settings Help	
10 🖻 🗃 🗿 🔦 🔌 🔏 🖻	👚 🕒 💁 🚱 👂 🗭 🐼 🛛 Build target: Debug 💦 📲 🖷 🖓 🗇 🐼	🗖 i.
:	✓ main(): int	-
Management X	main.cpp ×	⊲ ⊳
Projects Symbols 4 ► Workspace Sources Sources main.cpp	<pre>1 finclude <iostream> 2 3 using namespace std; 4 5 int main() 6 7 8 0 1 cout << "Hello world!" << endl return 0; 10</iostream></pre>	
		۰.
	Logs & others	x
	📝 Code::Blocks 🔍 Search results 🤤 Build log 📌 Build messages 🕥 Debugger 🔍 Thread search	4 Þ
	File Line Message	^
	D:\New folder\ In function `int main()':	
	D:\New folder\ 8 error: expected `;' before "return"	E
	=== Build finished: 1 errors, 0 warnings ===	-
	• m	- F
D:\New folder\Test\main.cpp	default Line 8, Column 1 Insert Read/Write default	

البته اگر خطا به ننوشتن ; مربوط باشد نرمافزار شماره خط خطا را خط بعد مشخص میکند.

دسته دیگر خطاهای این نوع، ننوشتن فایلهای کتابخانهای توابعی است که در برنامه استفاده شده است.



خطاهای منطقی

منظور از این نوع، خطاهایی است که در اجرای برنامه مشکلی ایجاد نمیکند و خروجی برنامه نیز ظاهر میشود اما جواب برنامه درست نیست. تشخیص این خطا به مراتب سختتر از خطای نوع اول است.

با محدود کردن اجرای برنامه و مشاهده مقادیر متغیرهای تعریف شده میتوانید خطا را شناسایی کنید.

برای دیدن مقدار متغیرها از منوی Debug و زیرمنوی Debugging windows گزینه را انتخاب کنید. در پنجره باز شده نام متغیرها را تایپ کنید و سپس با محدود کردن اجرا میتوانید مقادیر این متغیرها را مشاهده کنید و به این ترتیب محل بروز خطا را تشخیص دهید.

در قدم بعد باید محدوده اجرای برنامه را مشخص کرد. به این منظور از Breakpointها استفاده میشود. زمانی که برای دستوری Breakpoint مشخص شود برنامه با رسیدن به این دستور مکث میکند و وارد محیط دیباگ میشود. سپس میتوان از دستورات Step و سایر موارد استفاده کرد.

برای تعیین Breakpoint یک دستور میتوان از روشهای زیر استفاده کرد:

- ۱. بر روی خط دستور راست کلیک کرده و گزینه Toggle breakpoint را انتخاب کنید.
- ۲. بر روی خط دستور کلیک کرده و از منوی Debug گزینه Toggle breakpoint را انتخاب کنید.
 - ۳. بر روی خط دستور کلیک کرده و کلید F5 را فشار دهید.



در ابتدای خطی که برای آن Breakpoint تعیین میکنید یک دایره قرمز قرار میگیرد.



بعد از تعیین Breakpointها برای اجرای برنامه میتوانید از منوی Debug گزینه Start/Continue را انتخاب کنید یا از کلید F8 استفاده کنید. برنامه تا رسیدن به اولین Breakpoint اجرا میشود و صفحه خروجی نمایش داده میشود.

در ادامه موارد زیر به شما این امکان را میدهند که مراحل پیشروی دستورات را همان گونه که تمایل دارید انتخاب کنید:

- Debug این گزینه برنامه را خط به خط اجرا میکند. میتوانید از منوی Next line: ۱۰ انتخاب کنید یا کلید FV را فشار دهید
- Next این تفاوت که در Next اine هست با این تفاوت که در Next اine ine زمانی که با دستور فراخوانی یک تابع روبرو شود، وارد آن تابع نمی شود و مکان نما به خط بعدی منتقل می شود. اما در Step into اجرای برنامه به آن تابع منتقل می شود و مکان نما ابتدای اولین دستور تابع قرار می گیرد و بعد از اجرای کامل آن تابع به تابع فراخوانی کننده بر می گردد. این گزینه را می توانید از منوی Debug انتخاب کنید یا از کلیدهای ترکیبی Shift+F7 استفاده کنید.

 ۳. Step into این گزینه از جهت منطقی عملکردی بر عکس Step into دارد. اگر با اجرای Step into یا دلیل دیگری وارد یک تابع شده اید میتوانید با اجرای Step out از آن خارج شده و مکان نما به تابع فراخوانی کننده منتقل می شود. این گزینه را میتوانید از منوی Debug انتخاب کنید یا از کلیدهای ترکیبی Ctrl+F7 استفاده کنید.

برای حذف کردن Breakpoint میتوان روی آن کلیک کرد. همچنن برای حذف همه ها میتوان از منوی Debug گزینه Remove all breakpoints را انتخاب کرد.

گزینه دیگری غیر از Breakpoint نیز برای محدود کردن اجرا وجود دارد و آن Run to هست.

ابتدا مکان نما را روی خطی که قصد دارید عمل اجرا تا آن مکان انجام شود قرار دهید. سپس از منوی Debug گزینه Run to cursor را انتخاب کنید یا کلید F4 را فشار دهید. در آغاز خط مشخص شده، مثلث زرد رنگی قرار میگیرد و برنامه بلافاصله تا این دستور اجرا شده و سپس مکث میکند.

۲.۱ کامپایلرهای ++C روی گوشیهای هوشمند و تبلت

به منظور اجرای برنامههای ++C بر روی سیستمعامل اندروید، چندین نرمافزار وجود دارد که از جمله معروفترین و قویترین آنها C4droid هست. برای استفاده از این کامپایلر مراحل زیر را انجام دهید:

 ۱. به این آدرسهای زیر مراجعه کنید. هر سه فایل مربوط به گوشیهای هوشمند را بارگذاری کنید.

http://bayanbox.ir/download/7724964434733362302/ C4droid-CCpp-compiler-IDE-4.97.apk http://bayanbox.ir/download/220392741468215768/ GCC-plugin-for-C4droid-C-IDE-4.9.1.apk http://bayanbox.ir/download/6747071117453643935/

SDL-plugin-forC4droid.apk

- ۲. نرمافزار C4droid C/C++ compiler & IDE را نصب کنید.
- ۳. پلاگینهای GCC و SDL را نیز برای اجرای کامل برنامههای ++C نصب کنید.
- ۰۴. آیکون برنامه به شکل آ.۱آ است که در قسمت برنامههای کاربردی قابل مشاهده است.

برنامه را اجرا کنید. در کادر باز شده گزینه اول Install to internal memory (نصب در حافظه داخلی) را انتخاب کنید. در صورتی که گوشی روت شده باشد می توانید گزینه Install to external memory (نصب در حافظه خارجی) را کلیک کنید.



منتظر بمانید تا فایلهای پلاگین در حافظه وارد شوند.

۵. از منوی Preferences برنامه و در قسمت Select compiler گزینه ++ G را انتخاب کنید.





۱.۲.۱ نوشتن و اجرای برنامه

محیط برنامه شامل یک صفحه سفید و کلیدهای کنترلی است.

برای نوشتن توابع کتابخانهای برنامه دقت کنید که تابع iostream نیازی به پسوند h. ندارد.

شکل آ.۲



شکل آ.۳

بعد از اتمام دستورات و به منظور کامپایل برنامه دکمه Compile و جهت اجرای برنامه دکمه Run را انتخاب کنید.



خروجی این برنامه به شکل زیر است:

يبوست ب

خطایابی در Code::Blocks

در این بخش به بررسی خطایابی برنامههای ++C در محیط Code::Blocks خواهیم پرداخت. اما قبل از آن لازم است تا اندکی با مقدمات خطایابی آشنا شویم. لازم به ذکر است که خطایابی بیشتر یک هنر است تا یک علم، اما برای انجام صحیح آن لازم است تا با برخی از الگوهایی که در بین برنامهنویسان رواج دارد، آشنا شویم. به صورت کلی، خطایابی برنامهها به سه قسمت قابل تقسیم است:

- خطایابی در مرحله الگوریتم و فلوچارت، شامل رفع ایرادات منطقی در روش حل مسئله مورد نظر است. به عبارت دیگر، منظور از خطایابی در این مرحله، بدست آوردن یک الگوریتم مناسب و صحیح برای مسئله مورد نظر است. در این مرحله، لازم است تا با استفاده از ابزارهای ریاضی نشان دهیم که یک الگوریتم به درستی مسئله مورد نظر را حل میکند.
- ۲. رفع ایرادات نحوی برنامه که در این مرحله، لازم است تا تمام خطاهایی که توسط مترجم ++C مشخص شدهاند، رفع شوند تا مترجم برنامه قابل اجرا را تولید کند.
- ۳. پس از اینکه فایل اجرایی برنامه به صورت موفقیت آمیز توسط مترجم تولید شد، نوبت به خطایابی منطقی میرسد. ممکن است پیادهسازی الگوریتم در مرحله تبدیل الگوریتم به کد به درستی صورت نگرفته باشد و یا حتی نیاز به خطایابی در مرحله

الگوريتم و فلوچارت داشته باشيم.

خطایابی منطقی یکی از دشوارترین مراحل خطایابی است. این مرحله از خطایابی، یکی از پرهزینهترین مراحل برنامهسازی است. یکی از سادهترین و در عین حال پراستفادهترین راهکارهای خطایابی منطقی، آزمودن برنامه در برابر تعداد مشخصی ورودی و خروجی آزمون است. به عبارت دیگر، داده آزمون برای یک برنامه عبارت است از تعدادی مشخص ورودی به همراه خروجیهای مرتبط. در صورتی که برنامه حتی به ازای یک مورد از ورودیهای آزمون، خروجی مناسب تولید نکند، میتوان مطمئن بود که برنامه دارای خطای منطقی است. اولین و مهمترین اصل در خطایابی عبارت است از

«فرایند خطایابی یک برنامه عبارت است از بررسی تمام قسمتهای برنامه که فکر میکنیم صحیح هستند تا از صحیح بودن آنها اطمینان حاصل شود. هنگامی که به نادرستی یک قسمت از برنامه پیبردیم، میتوان گفت که شواهدی از وجود و محل خطا بهدست آمده است.»

نکته مهم در خطایابی این است که هنگامی که به یک خطا رسیدیم، نباید ناامید شویم! درست است که یک برنامه ایدهآل باید بدون خطا باشد، اما در عمل رسیدن به چنین ایدهآلی در عمل، اگر نگوییم ناممکن، دشوار است. هنگامی که شما خطای برنامه خود را فهمیدهاید، در راستای تصحیح آن تلاش میکنید و این نکته، خود ارزشمند است.

سادهترین و معمولترین شیوه برای خطایابی برنامهها، اضافه کردن عبارتهای متوالی ...» cout در طی برنامه است. به چنین قطعه کدهایی، قطعه کدهای رهگیر گفته می شود. در ادامه قصد داریم تا به روشهای خطایابی با برررسی یک پیادهسازی از الگوریتم مرتبسازی در جرمی بپردازیم. دلیل اینکه در این بخش از الگوریتمی استفاده شده است که شرح آن در کتاب نیامده، تمرکز بر روشهای خطایابی با استفاده از خطایابها است. بنابراین قطعه کد زیر را مشاهده کنید.

1 #include <iostream>
2 #include < cstdlib >
3 using namespace std;

```
4 void ins (int):
 5 void process ();
 6 void jump(int);
7 int x[10], // input data
       y[10]. // working space
8
9
       numlnputs = 0, // how many numbers to sort,
10
      indexY = 0; // how many elements are in y
  int main(int argc, char ** argv) {
H.
12
       // getting input data from the command line
13
       numlnputs = argc - I;
14
       for (int i = 0; i < numlnputs && i < 10; ++i) {
15
           x[i] = atoi(argv[i + 1]);
16
       }
17
       for (int i = 0; i < numlnputs; ++i) {
           cout << x[i] << "\t";
18
19
       }
20
       cout << endl:
21
       process();
22
       for (int i = 0; i < numlnputs; ++i) {
23
           cout << y[i] << "\t";
24
       }
25
       cout << endl:
26
       return 0:
27 }
28 void jump(int j) {
29
       for (int k = indexY - I; k > j; ++k) {
           y[k] = y[k-1];
30
31
       }
32 }
33 void ins (int newY) {
34
       if (indexY == 0) {
35
           y[0] = newY;
36
           return:
37
       }
38
       else {
39
           for (int j = 0; j < indexY; ++j) {
40
                if (newY < y[i])
```

```
41
                    jump(j);
42
                    y[i] = newY;
43
                    return:
44
                }
45
            }
        }
46
47 }
48 void process () {
        for (indexY = 0; indexY < 10 && indexY < numInputs; ++indexY) {
49
            ins (x[indexY]):
50
51
        }
52 }
همانگونه که مشاهده میکنند، این برنامه فاقد هرگونه خطای نجوی است و به سادگی
به کد اجرابی تبدیل می شود. اما در صورتی که آن را با داده آزمون فراخوانی کنیم، نتیجه
            برخلاف انتظار خواهد بود. برای مثال، قصد داریم تا این برنامه را با ورودی
                    8° 1° 4° 10 1° 14 19 8
بیازماییم. به همین دلیل، از منوی Project روی گزینه Set Programs' Arguments کلیک
کرده و سیس در کادر اول، گزینه Debug را انتخاب کرده و در کادر Program arguments
داده مورد نظر را به صورت فوق (اعداد با یک فاصله از هم جدا شدهاند) وارد میکنیم و روی
                        دکمه OK کلیک میکنیم (شکل ب.۱). خروجی مورد انتظار،
                   8° 1° 1° 1° 14 19 8
                    8 14 10 19 To to So No
                             است اما خروحی تولید شده به صورت زیر خواهد بود:
               8. 1. 4. 10 1. 14 19 8
               Segmentation fault (core dumped)
خطا کجا است؟ برای خطایابی، میتوان از کدهای رهگیر استفاده کرد. از خروجی و مقایسه
```

حص حب است : برای حصیابی، شی وال از عناقالی رفتین استفاده نود از خط شماره ۲۶ است، زیرا آن با متن برنامه متوجه می شویم خطا هر جا که هست، بعد از خط شماره ۲۶ است، زیرا خطوط قبلی به درستی اجرا شده اند (می توانید توضیح دهید چرا؟)

Select target	<u> </u>
Debug	
Release	
This target provides the project's main ex	ecutable
Program arguments:	
60 80 40 15 20 14 19 6	^
	-
Host application:	
	B
Run host in terminal	
OK Cancel	

شکل ب.۱: تنظیم آرگومانهای ارسالی به برنامه.

به همین دلیل، در اولین تلاش سعی میکنیم از اجرای صحیح دستورات مطمئن شویم. به همین منظور میتوان پس از هر خط، یک دستور

که lineNo بیانگر شماره خط برنامه مورد نظر است، اضافه میکنیم. با اضافه کردن این دستور پس از خط ۲۸ و ترجمه و اجرای مجدد برنامه، متوجه می شویم که خطا در این خط رخ می دهد، زیرا نتیجه اجرای کد رهگیر مربوط به این خط در خروجی مشخص نمی شود. به همین دلیل، به داخل بدنه تابع process رفته و در آن کدهای رهگیر را قرار می دهیم. { void process ()

62 for (indexY = 0; indexY < 10 && indexY < numInputs; ++indexY) {</p>
63 ins (x[indexY]);

64 cout << "Command at line " << 64 << " was run successfully

```
!" << endl
65
       }
66 }
با اجرای برنامه، متوجه میشویم در تکرار سوم حلقه، در اجرای خط ۶۴ خطا رخ میدهد.
60 80 40 15 20 14 19 6
Command at line 64 was run successfullv!
Command at line 64 was run successfully!
Segmentation fault (core dumped)
شاید دانستن مقدار متغیر indexY و متغیر numInputs که سانگر تعداد پارامترهای
ورودی به برنامه است، به ما در تشخیص محل خطا کمک کند. به همین دلیل، تابع را به
                                      این فرم به کدهای رهگیر مجهز میکنیم.
61 void process () {
       cout << "The numInputs = " << numInputs << endl;</pre>
62
       for (indexY = 0; indexY < 10 && indexY < numlnputs; ++indexY) {
63
64
           ins (x[indexY]);
65
           cout << "Command at line " << 64 << " was run successfully
                with indexY = " << index Y << endl;
66
       }
67 }
                                       و خروجی حاصل به فرم زیر است
60 80 40 15 20 14 19 6
The numInputs = 8
Command at line 64 was run successfully with indexY = 0
Command at line 64 was run successfully with indexY = 1
Segmentation fault (core dumped)
```

بنابراین به تابع ins مشکوک میشویم و آن را به کدهای رهگیر زیر مجهز میکنیم

```
44 void ins (int newY) {
       cout << "newY = " << newY << endl:
45
       if (indexY == 0) {
46
47
          y[0] = newY;
          cout << "Line 48 run sucessfully !" << endl:
48
49
          return:
50
       }
51
       else {
          for (int j = 0; j < indexY; ++j) {
52
              cout << "j = " << j << endl;
53
54
               if (newY < y[j]) 
55
                  iump(i):
56
                  cout << "Line 57 run sucessfully !" << endl;
57
                  y[i] = newY;
                  cout << "Line 58 run sucessfully !" << endl;
58
59
                  return:
60
              }
          }
61
62
       }
63 }
                  و خروحی حاصل از احرای برنامه به صورت زیر خواهد بود.
60 80 40 15 20 14 19 6
The numInputs = 8
newY = 60 and indexY = 0
Line 48 run sucessfully!
Command at line 64 was run successfully with indexY = 0
newY = 80 and indexY = 1
j = 0
Command at line 64 was run successfully with indexY = 1
newY = 40 and indexY = 2
j = 0
```

Segmentation fault (core dumped)

```
با بررسی خروجی، میتوان متوجه شد که خطا در خط ۵۶ به ازای ;jump(j) رخ میدهد.
                                       به همین دلیل، تابع jump را مورد بررسی قرار میدهیم.
38 void jump(int j) {
           cout \langle \langle i \rangle = \langle \langle i \rangle \langle \langle i \rangle \rangle and indexY = \langle \langle i \rangle \langle \langle i \rangle \rangle \langle \langle i \rangle \rangle endl;
39
           for (int k = indexY - I; k > j; ++k) {
40
                cout << "k = " << k << endl;
41
                y[k] = y[k-1];
42
                cout << "Line 43 run successfully !" << endl:
43
44
           }
45 }
                                 خروجی حاصل از اجرای این برنامه به صورت زیر است
```

```
60 80 40 15 20 14 19 6
The numInputs = 8
newY = 60 and indexY = 0
Line 48 run sucessfully!
Command at line 64 was run successfully with indexY = 0
newY = 80 and indexY = 1
j = 0
Command at line 64 was run successfully with indexY = 1
newY = 40 and indexY = 2
j = 0
j = 0 and indexY = 2
k = 1
Line 43 run successfully!
k = 2
```

Line 43 run successfully! . . . k = 902Line 43 run successfully! k = 903Line 43 run successfully! k = 904Segmentation fault (core dumped) با کمی بررسی دقیقتر متوجه میشویم که برنامه در صورت ورود به حلقه 40 for (int k = indexY - 1; k > j; ++k) { هیچگاه نمی تواند از این حلقه خارج شود. به عبارت دیگر، در این جا یک خطای يبادهسازي داريم! براي اينكه ابن حلقه خاتمه يبدا كند، با بابد شرط ادامه حلقه را تغيير دهيم و يا دستور ادامه حلقه را تصحيح كنيم. اين حلقه را به صورت زير تصحيح ميكنيم 40 for (int k = indexY - I; k > j; --k) { یس از ترجمه مجدد برنامه، خروجی برنامه به صورت زیر خواهد بود 60 80 40 15 20 14 19 6 The numInputs = 8newY = 60 and indexY = 0Line 48 run sucessfully! Command at line 64 was run successfully with indexY = 0newY = 80 and indexY = 1j = 0 Command at line 64 was run successfully with indexY = 1 newY = 40 and indexY = 2

```
i = 0
i = 0 and indexY = 2
k = 1
Line 43 run successfully!
Line 57 run sucessfully!
Line 58 run sucessfully!
Command at line 64 was run successfully with indexY = 2
newY = 15 and indexY = 3
i = 0
j = 0 and indexY = 3
. . .
k = 1
Line 43 run successfully!
Line 57 run sucessfully!
Line 58 run sucessfully!
Command at line 64 was run successfully with indexY = 7
Command at line 28 was run successfully!
6 14 15 19 20 40 60 0
```

با اینکه اجرای برنامه به درستی صورت پذیرفت، اما هنوز یک مورد باقی مانده است. تعداد اعداد ورودی ۸ مورد است، اما در خروجی تنها ۷ مورد نمایش داده شدهاند و بالاترین مقدار نیز نمایش داده نشده است. قبل از اینکه بخواهیم به خطایابی این ایراد بپردازیم، لازم است تا کدهای رهگیر مربوط به خطای قبلی را غیر فعال کنیم. این امر فرایندی زمانبر است، به ویژه برای مواردی که تعداد خطاها بیشتر یا پیچیدگی بیشتری داشته باشند. به همین منظور، لازم است تا یک روش مناسب برای خطایابی ابداع کنیم. اولین چیزی که به ذهن میرسد، برنامهای است که خط به خط برنامه مورد نظر را اجرا کند و پس از اجرای هر خط، مقادیر متغیر مورد نیاز را به ما نشان دهد. شاید بهتر باشد، اجرای برنامه را تا خط مشخصی ادامه داده و در آن خط برای وارسی مقادیر متغیرهای برنامه متوقف شود و پس از بررسی، اجرای برنامه را از سر گیرد.

خوشبختانه چنین برنامهای وجود دارد که به آن خطایاب یا Debugger گفته می شود. یکی از خطایاب های بسیار قدرتمند و البته متن باز که در Code::Blocks نیز استفاده می شود، GDB یا GNU DeBugger است. قبل از ادامه خطایابی، لازم است تا کدهای رهگیر اضافه شده را حذف کنیم. بدین منظور می توانید به ابتدا هر خط از دستورات رهگیر، نشان توضیح یا // را اضافه کنید و یا آن خط را حذف کنید. در ادامه، ما فرض می کنیم که شما کدهای رهگیر را حذف کرده اید.

با توجه به تقسیمبندی منطقی برنامه، احتمال میدهیم خطا در بدنه تابع process باشد. برای بررسی عملکرد این تابع بدون اضافه کردن کدهای رهگیر، از خطایاب استفاده میکنیم. برای اینکه برنامه تولید شده قابلیت خطایابی داشته باشد، لازم است تا آن را با استفاده از کلید g- و بدون کلید s- ترجمه کنیم. بدین منظور، از منوی Project، گزینه ... Build options را انتخاب میکنیم. لازم است تا در این پنجره پس از انتخاب بخش Debug، شکل ب.۲، گزینههای مرتبط با کلید g- و Wall- فعال باشند.

پس از اعمال این تنظیمات، از منوی Build، بخش Select target، گزینه Debug، گزینه Debug را انتخاب میکنیم. حال زمان آن رسیده است تا یک بار دیگر برنامه را به کد اجرایی تبدیل کنیم. بدین منظور از منوی Build گزینه Rebuild را انتخاب میکنیم تا خروجی مورد نظر تولید شود. حال میتوان برنامه را با استفاده از GDB خطایابی کرد. برای اجرای برنامه در حالت خطایاب، لازم است تا از منوی Debug گزینه Start / Continue را انتخاب کنیم. برنامه مانند سابق اجرا میشود. برای اینکه اجرای برنامه را در خط خاصی متوقف کنیم، لازم است تا از نقاط توقف یا breakpoint استفاده کنیم. برای اضافه کردن یک نقطه توقف در روش که ساده ترین است، راست کلیک کردن بر روی شماره خط در کد منبع و انتخاب گزینه روش که ساده ترین است. پس از انتخاب این گزینه، یک دایره قرمز رنگ در کنار شماره خط

Project build	options				
sample	Selected compiler				
Release	GNU GCC Compiler				
Keicube	Compiler settings Linker settings Search directories Pre/post build steps Custom variables				
	Policy: Append target options to project options				
	Compiler Flags Other options #defines				
	Categories: <all categories=""></all>				
	Produce debugging symbols [-g] Profile code when executed [-pg] In C mode, support all ISO C90 programs. In C++ mode, remove GNU extensions that cc Enable all compiler warnings (overrides many other settings) [-Wall]				
	Enable extra compiler warnings [-Wextra] Stop compiling after first error [-Wfatal-errors] Tubihit all warning messages [-w]				
	Have g++ follow the 1998 ISO C++ language standard [-std=c++98] Have g++ follow the coming C++0x ISO C++ language standard [-std=c++0x]				
	Have g++ follow the C++11 ISO C++ language standard [-std=c++11] Enable warnings demanded by strict ISO C and ISO C++ [-pedantic] Treat as errors the warnings demanded by strict ISO C and ISO C++ [-pedantic-errors]				
	OK Cancel				

شکل ب.۲: تنظیمات لازم برای تولید نسخه مناسب خطایابی.

ظاهر میشود (شکل ب.۳). روش دوم، قرار دادن نشانگر متن در خط مورد نظر و سپس



شکل ب.۳: اضافه شدن یک breakpoint

انتخاب گزینه Toggle breakpoint از منوی Debug است. همچنین میتوان از کلید میانبر F5 نیز استفاده نمود. پس از اضافه نمودن نقطه توقف در خط شماره ۲۳ و اجرای برنامه در حالت خطایابی (منوی Debug، گزینه Start / Continue)، برنامه اجرا شده اما اجرای آن قبل از خط شماره ۲۳ متوقف میشود. با انتخاب گزینه Watches از منوی Debug بخش Debugging windows، میتوان دیدبانهای مختلف را مشاهده نمود. وظیفه دیدبان، نمایش مقدار متغیری است که وظیفه دیدبانی از آن را به عهده دارد. برای اضافه کردن دیدبان، کافی است نام متغیر را انتخاب کرده و روی آن راست کلیک کرده و از منوی کشویی، گزینه x' Watch را انتخاب کنیم (x' نام متغیر است). قبل از اینکه بخواهیم وارد خطایابی تابع Process شویم، شاید بد نباشد به مقادیر نگهداری شده در آرایه x نگاهی بیاندازیم. بدین منظور، یک دیدبان برای x قرار میدهیم. به محض انجام این کار، مقدار متغیر x در پنجره دیدبانی نمایش داده می شود (شکل ب.۴). با دقت در دادههای نمایش داده شده در پنجره

Wa	Watches (new)				
	argc	9	int		
Œ	argv		int [10]		
Œ	х		int [10]		

شکل ب.۴: پنجره دیدبانی

دیدبانی، مشاهده میشود که متغیر x یک آرایه از نوع داده int به طول ۱۰ است. با کلیک بر روی علامت + در کنار نام متغیر x در پنجره دیدبانی، مقادیر مختلف هر یک از خانههای آرایه x نمایش داده میشوند (شکل ب۵). با مشاهده مقادیر x میتوانیم اطمینان حاصل کنیم که همه مقادیر ورودی به درستی در آرایه x قرار گرفتهاند.

بنابراین مسیر خطایابی را ادامه میدهیم. اگر بر روی گزینه Start / Continue کلیک کنیم، اجرای برنامه تا پایان آن ادامه مییابد. اما قصد ما ورود به بدنه تابعی است که نقطه توقف در آن قرار دارد. به همین دلیل، از گزینه Step into استفاده میکنیم تا اجرای برنامه وارد بدنه تابع process شده و در ابتدای تعریف تابع متوقف شود. هنگامی که خطایاب اجرای برنامه در یک خط را متوقف میکند، در کنار آن خط یک نشانگر زردرنگ قرار میگیرد (شکل ب.۶). برای اجرای خط به خط برنامه از محل توقف جاری، از گزینه Next line از منوی Debug استفاده میکنیم. با بررسی شرایط حلقه for در بدنه تابع step میتوان متوجه شد که ایرادی از نظر منطقی در این قسمت از برنامه قرار ندارد و خطا در بدنه تابع ins است. برای ورود به بدنه تابع متوان یک نقطه توقف در ابتدای بدنه تابع step قرار

Watches (new)			
	argc	9	int
Ð	argv		int [10]
	x		int [10]
	[0]	60	
	[1]	80	
	[2]	40	
	[3]	15	
	[4]	20	
	[5]	14	
	[6]	19	
	[7]	6	
	[8]	0	
	[9]	0	

شکل ب.۵: نمایش مقادیر آرایه x در پنجره دیدبانی

شکل ب.۶: وجود نشانگر زرد رنگ در کنار شماره خط بیانگر توقف خطایاب در آن خط است.

داد و یا از دستور Step into استفاده نمود. در اینجا ما از روش اول استفاده میکنیم، یعنی یک نقطه توقف در خط شماره ۴۰ قرار میدهیم و دستور Continue / Start / Continue را از منوی Debug انتخاب میکنیم. از آنجا که نتیجه مرتب سازی در آرایه y قرار میگیرد، لازم است تا برای متغیر y نیز یک دیدبان قرار دهیم. اجرا را از خط شاره ۴۰ به بعد به صورت خط به خط ادامه میدهیم. در هنگام بررسی متغیرها از طریق دیدبان متوجه میشویم که هنگامی که در تکرار دوم از حلق می در بنا تابع signa از طریق دیدبان متوجه میشویم که هنگامی می در تارا دوم از حلقه خط ادامه میدهیم. در هنگام بررسی متغیرها از طریق دیدبان متوجه میشویم که هنگامی می در تکرار دوم از حلقه خط ۵۷ در بدنه تابع process تابع signa با آرگومان ۸۰ فراخوانی میشود، مقدار ۸۰ در آرایه y قرار نمیگیرد و برنامه به سراغ عنصر بعدی میرود. برای خطایابی، لازم است تا مقدار متغیر ۲ نور نمیگیرد و برنامه به سراغ عنصر بعدی میرود. برای خطایابی، لازم است تا مقدار متغیره با مشاهده دیدبانها متوجه میشویم که در صورتی خط دطایابی، لازم است تا مقدار میکنیم. با مشاهده دیدبانها متوجه میشویم که در صورتی که عنصر جاری از هری از می می در اختیار داشته باشیم، به همین دلیل، می دیدبان برای آن نیز اضافه میکنیم. با مشاهده دیدبانها متوجه میشویم که در صورتی که عنصر جاری از همه عناصر آرایه y قرار بگیرد. بدین منظور، پس از پایان حلقه for در انتها است تا در انتهای آرایه و قرار بگیرد. بدین منظور، پس از پایان حلقه موا در خط

شماره ۵۲، دستور زیر را اضافه میکنیم

52 y[indexY] = newY;

اما متاسفانه هنوز برنامه به درستی کار نمیکند! به همین دلیل، خطایابی را مجددا در تابع ins ادامه میدهیم. با ادامه اجرای برنامه در حالت خطایاب، از طریق بررسی دیدبانها متوجه میشویم که به محض اولین اجرای تابع jump، عنصر ۸۰ از آرایه y حذف میشود! بنابراین لازم است در اجرای تابع jump دقت بیشتری کنیم!

به همین منظور، خط شماره ۳۴ را با یک نقطه توقف علامتگذاری میکنیم و برنامه را در حالت خطایاب اجرا میکنیم. با استفاده از دیدبانها، ایراد برنامه به سادگی مشخص میشود! ایراد در مقدار اولیه متغیر k در حلقه for در خط ۳۴ است که باعث از بین رفتن مقدار [yindexY] میشود. به منظور تصحیح این خطا، کافی است کران پایین حلقه را به صورت زیر تصحیح کنیم.

34 for (int k = indexY; k > j; --k) {

با اعمال این تصحیح و اجرای برنامه، متوجه میشویم که خطای برنامه به ازای این ورودی رفع شده و نتیجه صحیح در خروجی نمایش داده میشود.

در این بخش، ما از خطایابها برای یافتن خطای برنامه در حین اجرا استفاده کردیم. یکی دیگر از مزایایی که خطایابها میتوانند داشته باشند، فهمیدن عملکرد برنامهها و الگوریتمها با استفاده از آنها است.