جلسه هجدهم – زهراسادات طباطبائی-25/9/99

# تنظیمات مربوط به کامپایل و دیباگ کردن

کامپایلر مجموعه‌ای از برنامه یا برنامه‌های کامپیوتری است که متنی از زبان برنامه‌نویسی سطح بالا (c)را به زبانی سطح پایین(مثل اسمبلی یا زبان سطح ماشین) تبدیل می‌کند. وقتی ما فایلی را کامپایل میکنیم در ئاقع یک فایل از نوع exe ایجاد می کنیم که قابل خواندن می شود .

و دیباگ کردن فرایندی است که در طی ان می شود ایراد های برنامه نوشته شده را گرفت .

## کامپایل کردن

کامپایل کردن به دو روش دستی و اتوماتیک انجام میشود که در ادامه به آن می پردازیم :

### کامپایل به روش اتوماتیک

Ctrl + shift + p

configure default build task

c/c++ : gcc.exe build active file

دقت کنید که ادرس کامپایلر درست باشد :

Compiler : c:\MinGW\bin\gcc.exe

این کار فایل task.json رو میسازد و بعضی کار ها را اتوماتیک انجام می دهد .

1. {
2. "version": "2.0.0",
3. "tasks": [
4. {
5. "type": "cppbuild",
6. "label": "C/C++: gcc.exe build active file",
7. "command": "C:\\MinGW\\bin\\gcc.exe",
8. "args": [
9. "-g",
10. "${file}",
11. "-o",
12. "${fileDirname}\\${fileBasenameNoExtension}.exe"
13. ],
14. "options": {
15. "cwd": "C:\\MinGW\\bin"
16. },
17. "problemMatcher": [
18. "$gcc"
19. ],
20. "group": {
21. "kind": "build",
22. "isDefault": true
23. },
24. "detail": "compiler: C:\\MinGW\\bin\\gcc.exe"
25. }
26. ]
27. }

شکل 1 – فایل tasks.json

این کار را با روش دیگری هم انجام می دهیم :

Ctrl + shift + b

Build task

c/c++ : gcc.exe build active file

* وقتی فایلی باز است هر وقت این دستور زا بزنیم همین فایلی که در آن هستیم را کامپایل می کند.
* باید دقت کنیم این کار درست انجام شود و ترمینال پیام زیر را نشان دهد و باید فایلی به اسم s20\_main.exe درست کرده باشد

Starting build …

Build finished successfully.

و حالا فابل ما کامپایل شده است .

### کامپایل به صورت دستی

Tasks.json بعضی کارها را اتوماتیک انجام می دهد ، بخش command یک سری پارامتر دارد :

-g

$(file) = فایلی که الان باز است

-o = اسم خروجی

$(filedirname) = اسم دیرکتوری که فایل در آن است

دستوری که می زند (داخل ترمینال) :

C:\ MinGW\bin >c:\MinGW\bin\gcc.exe -g C:\git\FC99001\class\s20\s20\_main.c -o C:\git\FC99001\class\s20\ s20\_main.exe

اگر بخواهیم دستی کامپایل کنیم باید این کد رو بزنیم .

## دیباگ کردن

### دستوراتی که باید بزنیم

Ctrl + shift + p

run task

c\c++ :gcc.exe build active file

یک break point میذاریم و F5 را میزنیم و C++ (GDB/LLDB) را انتخاب می کنیم وGcc.exe – build and debug active file را میزنیم :

Break point

F5

C++( GDB/LLDB)

Gcc.exe – build and debug active

این کار فایل launch.json را در پوشه .vscode درست میکند (tasks.json را در همان جا درست میکند)

برای دیباگ کردن حتما باید فایل exe , gdb.exe را داشته باشیم.

### تنظیمات توی فایل launch.json

برای مشاهده خروجی کار میتونیم در launch.json تغییراتی به وجود بیاریم:

"externalConsole": false,

این کار ترمینال جدایی را درست میکند تا بتوانیم خروجی کار را ببینیم .

### دیباگ کردن در cmd

دیباگر GDB در cmd اجرا میشود و کار با آن کمی سخت است ولی کار دیباگ را انجام میدهد، برای استفاده از آن مانند زیر عمل می کنیم

C:\git\FC99001\class\s20>gdb s20\_main.exe

اگر در مسیر قرارش داده باشیم :

(gdb) help

List of classes of commands:

aliases -- Aliases of other commands

breakpoints -- Making program stop at certain points

data -- Examining data

files -- Specifying and examining files

internals -- Maintenance commands

obscure -- Obscure features

running -- Running the program

stack -- Examining the stack

status -- Status inquiries

support -- Support facilities

tracepoints -- Tracing of program execution without stopping the program

user-defined -- User-defined commands

# ایجاد تابع و گرفتن خروجی

## برای نوشتن و خوندن در صفحه نمایش :

ما میتوانبم کتاب خانه رو import کنیم :

1. #include<stdio.h>

شکل 2 - include

برای هر کاری کتاب خانه خاصی را باید import کنیم

## تابع main

* همیشه باید فانکشنی به نام main داشته باشیم: (فعلا داخل پرانتز را میگذریم)

1. int main(int argc, char const \*argv[])
2. {
3. /\* code \*/
4. return 0;
5. }

شکل 3 – تابع main

اگه نخواهیم چیزی را ریترن کنیم void میگذاریم

## نوشتن توابع مورد نظر

* تابع هایی که میزنیم:

1. int sum(int a, int b)
2. {
3. return a + b;
4. }
5. void main(int argc, char const \*argv[])
6. {
7. Int c;
8. C = sum(5, 10);
9. Printf(“%d”,c);
10. }

شکل 4 – تابع sum

اگه نوع تابع void و چیزی اخرش پرینت می کرد درmain فقط تابع را صدا بزنیم کافیست .

### استفاده از for در تابع

* اولین پارامتری که میگیرد نقطه شروع ، دومی شرط ادامه حلقه و سومی این که چقدر بهش اضافه شود است

1. #include<stdio.h>
2. int print(int a)
3. {
4. for (int i = 0; i < a+1; i++)
5. {
6. printf("%d\n",i);
7. }
8. }
9. int main()
10. {
11. int a = 10;
12. print(a);
13. }

شکل 5 – تابع print

* خروجی تابع بالا :

0

1

2

3

4

5

6

7

8

9

10

شکل 6 – خروجی تابع print

### استفاده از if در تابع

1. #include<stdio.h>
2. int print(int a)
3. {
4. for (int i = 0; i < a; i++)
5. {
6. if(i%2==0)
7. {
8. printf("%d\n",i);
9. }
10. }
11. }
12. int main()
13. {
14. int a = 10;
15. print(a);
16. }

شکل 7 – استفاده از if در تابع print

* خروجی :

0

2

4

6

8

شکل 8 – خروجی تابع بالا

# تکلیف جلسه بعد

* نوشتن تابع چاپ اعداد اول 1 تا 100