

算法与数据结构设计报告

（ 2019 / 2020学年 第 一 学期）

题 目：

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **学 生 姓 名** |  |
| **班 级 学 号** |  |
| **指 导 教 师** |  |
| **指 导 单 位** | **计算机科学与技术系** |
| **日 期** | **2019.10.21-2019.11.1** |

|  |  |  |
| --- | --- | --- |
| **支撑指标点** | **评价准则** | **计分（每项10分）** |
| 课程目标1：文献调研与资料收集能力，问题发现、研究、分析与解决能力（20分） | 1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，收集资料进行合理的分析与设计 |  |
| 2、通过调研，能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现 |  |
| 课程目标2：通过课程设计，培养学生综合应用算法和数据结构等知识解决工程问题的实践能力（20分） | 3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述 |  |
| 4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便 |  |
| 课程目标3：培养解决工程问题的开发工具运用能力，能够利用程序设计软件或系统对问题求解进行模拟和实现，能够设计测试数据验证问题解决方法的正确性，并能够对问题解决方法的性能和效率进行分析（40分） | 5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标 |  |
| 6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解 |  |
| 7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能 |  |
| 8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试 |  |
| 课程目标4：选择同类课题的学生能够通过讨论和交流解决课程设计中的难题，能在实验报告中准确阐述课程设计的内容，能够清晰陈述观点和回答问题（20分） | 9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握 |  |
| 10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范 |  |
| 算法与数据结构设计能力测评总分 | |  |
| **指导教师： 年 月 日** | | |
| **备注：** | | |

**算法设计题1：C语言程序中函数间信息传递**

1. **课题内容和要求**

本课题的目标系统“C语言函数信息传递分析系统”的功能框架图如图1所示：

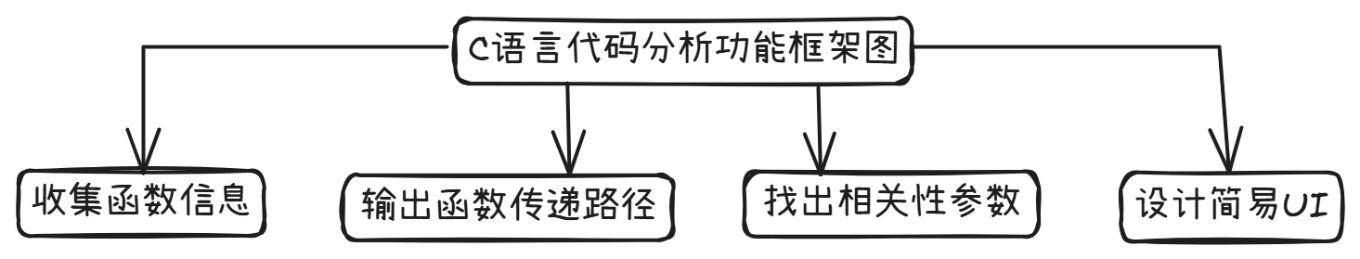


图1 功能框架图

1. 收集函数信息：通过字符串匹配或正则表达式对C语言程序中的函数调用进行识别，收集函数信息（函数名、函数的形参名和类型）；
2. 输出函数传递路径：得到并输出C语言程序中函数传递的路径（考虑到if else的情况）；
3. 找出相关性参数：找出函数调用路径间具有相关性的参数（如：functionA()返回值作为functionB()的输入，暂时不考虑返回值为函数的情况）；
4. 设计建议UI：有个简单直观的界面，以便展示上述功能。
5. **数据结构说明**

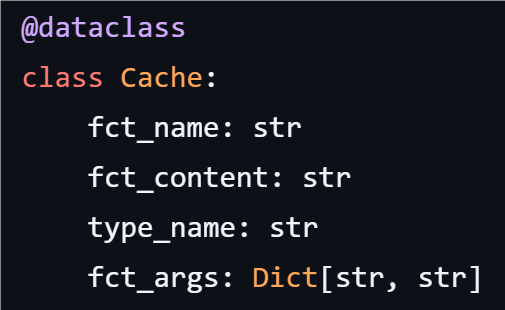


图2 存储函数信息的cache

如图2，Cache类用于存储函数的信息，其中：

fct\_name：函数名

fct\_content：函数内容

type\_name：函数类型

fct\_args：函数参数信息

**三、算法设计**

该算法的核心逻辑如图3：

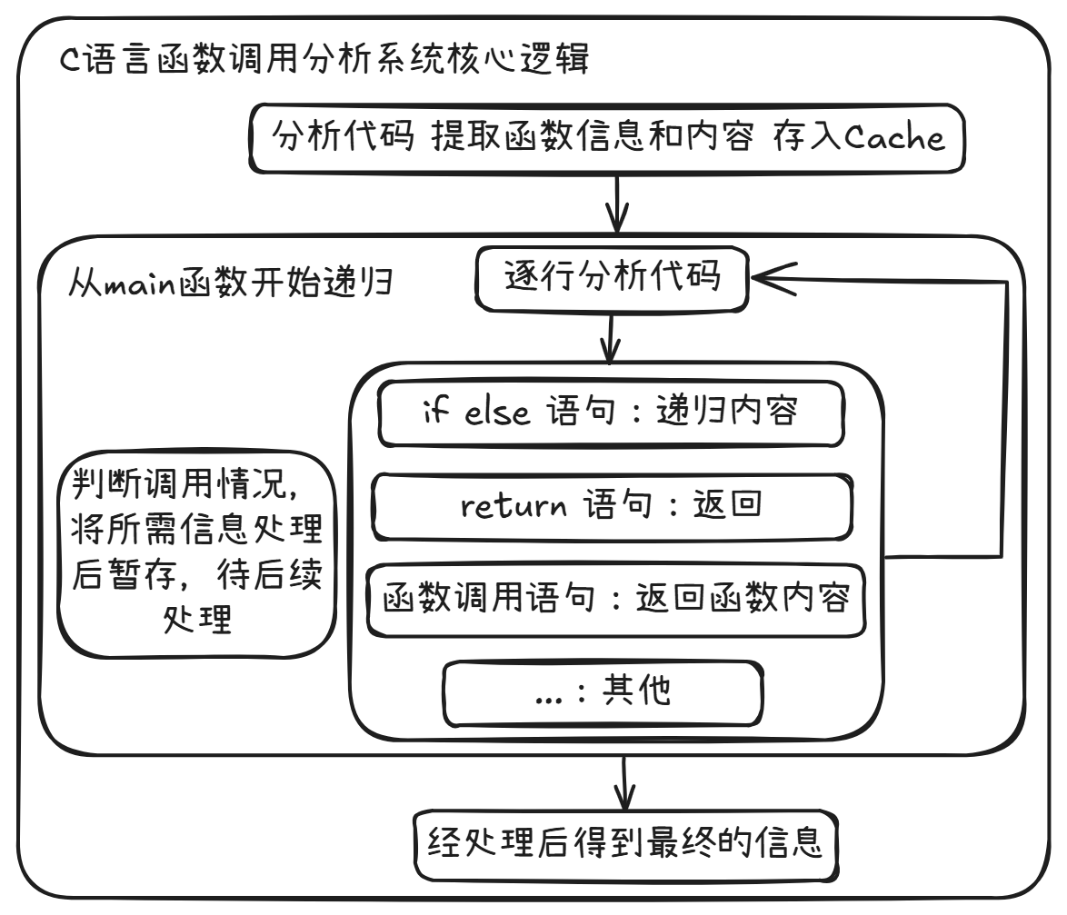


图3 核心代码逻辑

1分析代码：

（1）遍历所有C语言源代码，通过正则表达式识别函数并将其相关信息存入Cache中。

（2）逐行读取代码内容，利用正则表达式判断代码功能，再进行进一步分析。

2核心递归逻辑：

动态规划，根据需求设计出状态转移方程：dps(i+1，调用路径，C代码内容，条件) = dps(i，调用路径+[当前调用路径]，[所需代码内容]，条件+[当前条件])，据此设计出递归函数，从而遍历分析C代码内容。

3函数信息输入输出及更新处理逻辑：

单独设计一个class用于管理Cache存储内容，用法与字典类似。维护数据安全与长度，提高可变性。

4语言切换功能逻辑：

设计两个语言存储class（或者也可使用字典），将输入输出信息变量化，从而使得中英切换容易实现。

5最终输出逻辑：

通过设计一个新的class来维护最终的输出，转译源数据存储格式为所需的输出结果，并通过与设计的UI交互达到目的。

6 UI设计逻辑：

使用python自带的tkinter实现的一个建议GUI，通过类管理，提高稳定性。

**四、详细设计**

(1)核心递归逻辑：

def \_inner\_parse(self) -> None:

main\_cache: Cache = self.fct\_cache["main"]

if not main\_cache:

raise RuntimeError("`main` function not found.")

def dps(fct\_path: str, content: str, conditions: str) -> None:

for match in self.combined\_pattern.finditer(content):

if match.group(1): # if block

if\_block = match.group(1)

logger.debug(f"if block: {if\_block.strip()}")

if\_block = shrink\_code(str(if\_block))

condition = self.condition\_pattern.search(if\_block)

condition = condition.group(1) if condition else None

if\_body = self.extract\_if\_block(if\_block)

logger.debug(f"if body: {if\_body}")

dps(fct\_path, if\_body, f"{conditions} and {condition}")

continue

if match.group(2): # else block

else\_block = match.group(2)

logger.debug(f"else block: {else\_block.strip()}")

else\_block = shrink\_code(str(else\_block))

condition = self.condition\_pattern.search(if\_block) if if\_block is not None else None

condition = condition.group(1) if condition else None

else\_body = self.extract\_else\_block(else\_block)

logger.debug(f"else body: {else\_body}")

dps(fct\_path, else\_body, f"{conditions} and not {condition}")

continue

if match.group(3): # assign function, occupies 3 groups

assign\_block = match.group(3)

logger.debug(f"assign func: {assign\_block.strip()}")

assign\_match = re.findall(self.patterns["assign\_fct"], assign\_block)

if assign\_match:

val, func\_name = assign\_match[0]

logger.debug(f"assign function: {val} from {func\_name}")

cur\_fct\_name = fct\_path.split(" -> ")[-1]

self.relatives\_lst.append(f"<{cur\_fct\_name}(), {val}> = <{func\_name}(), return value>")

# call function

call\_fct\_cache = self.fct\_cache[func\_name]

if call\_fct\_cache is None:

logger.warning(f"Failed to find function state `{func\_name}`.")

else:

dps(fct\_path + " -> " + func\_name, call\_fct\_cache.fct\_content, conditions)

continue

if match.group(6): # call function, occupies 3 groups

call\_block = match.group(6)

logger.debug(f"call func: {call\_block.strip()}")

call\_match = re.findall(self.patterns["call\_fct"], call\_block)

if call\_match:

call\_fct\_name, var = call\_match[0]

logger.debug(f"call function: {call\_fct\_name}")

call\_fct\_cache = self.fct\_cache[call\_fct\_name]

if call\_fct\_cache is None:

logger.warning(f"Failed to find function state `{call\_fct\_name}`.")

else:

# fast replace

if call\_fct\_cache.fct\_args != {}:

arg = list(call\_fct\_cache.fct\_args.keys())[0]

logger.debug(f"args: var - {var} arg - {arg}")

self.\_fast\_replace(call\_fct\_cache.fct\_name, var, arg)

logger.debug(f"replaced content preview: {call\_fct\_cache.fct\_content}")

dps(fct\_path + " -> " + call\_fct\_name, call\_fct\_cache.fct\_content, conditions)

continue

# return value or none

if match.group(9):

norm\_block = match.group(9)

logger.debug(f"statement: {norm\_block.strip()}")

return\_match = re.search(self.return\_pattern, norm\_block)

if not return\_match:

continue

return\_value = return\_match.group(1)

logger.debug(f"return: {return\_value}\n")

# append result

self.walkthrough\_lst.append(fct\_path.split(" -> "))

self.conditions\_lst.append(conditions)

return

else:

logger.warning(f"Failed to classify code: {match}")

# end for function content

self.walkthrough\_lst.append(fct\_path.split(" -> "))

self.conditions\_lst.append(conditions)

return

dps("main", main\_cache.fct\_content, "")

（2）函数信息管理：

class FunctionStateCache:

def \_\_init\_\_(self):

self.caches: Dict[str, Cache] = {}

def \_\_len\_\_(self) -> int:

return len(self.cache)

def \_\_getitem\_\_(self, key: str) -> Cache:

return self.caches[key] if key in self.caches else None

def \_\_iter\_\_(self):

return iter(self.caches.values())

def update(

self,

fct\_name: str,

fct\_content: Optional[str] = None,

type\_name: Optional[str] = None,

fct\_args: Optional[Dict[str, str]] = None,

) -> None:

if fct\_name in self.caches:

cache: Cache = self.caches[fct\_name]

cache.fct\_content = fct\_content if fct\_content is not None else cache.fct\_content

cache.type\_name = type\_name if type\_name is not None else cache.type\_name

if fct\_args is not None:

cache.fct\_args.update(fct\_args)

else:

self.caches.update(

{

fct\_name: Cache(

fct\_name,

fct\_content,

type\_name,

fct\_args,

)

}

)

def \_output\_(self) -> None:

for cache in self.caches.values():

print(f"Function Name: {getattr(cache, 'fct\_name', None)}")

print(f"Return Type: {getattr(cache, 'type\_name', None)}")

print(f"Arguments: {getattr(cache, 'fct\_args', None)}")

print(f"Conditions: {getattr(cache, 'conditions', None)}")

print(f"Content: {getattr(cache, 'fct\_content', None)}")

print()

（2）UI交互核心：

class Core:

def \_\_init\_\_(

self,

DOCS: Union[DOCS\_CN, DOCS\_EN],

src\_code: Optional[Any] = None,

tk\_inputs: Optional[Any] = None,

tk\_outputs: Optional[Any] = None,

output\_fct\_cache: Optional[Any] = None,

output\_walkthroughs: Optional[Any] = None,

output\_relatives: Optional[Any] = None,

) -> None:

self.DOCS = DOCS

self.src\_code = tk\_inputs.get("1.0", tk.END) if src\_code is None else src\_code

logger.debug(f"src\_code: {self.src\_code}")

if self.src\_code is None or self.src\_code == "" or self.src\_code == "\n":

logger.error(f"`code` got None ({self.src\_code}).")

raise ValueError(f"`code` got None ({self.src\_code}).")

self.output\_fct\_cache = output\_fct\_cache.get()

self.output\_walkthroughs = output\_walkthroughs.get()

self.output\_relatives = output\_relatives.get()

self.tk\_outputs = tk\_outputs

self.pwd = os.getcwd()

# process

self.caches = FunctionStateCache()

self.parser = CodeParser(self.src\_code, self.caches)

self.\_output\_()

def \_output\_(self) -> None:

self.tk\_outputs.configure(state="normal")

self.tk\_outputs.delete("1.0", tk.END)

outputs = self.\_preprocessing\_before\_output()

outputs = "\n".join(outputs)

# TODO

self.tk\_outputs.insert(tk.END, outputs)

self.tk\_outputs.configure(state="disable")

def \_format\_fct\_statements(self) -> List[str]:

fct\_state\_caches = self.parser.fct\_cache

fct\_state\_lst = []

for cache in fct\_state\_caches:

args = (

", ".join([f"{dtype} {var}" for var, dtype in cache.fct\_args.items()])

if cache.fct\_args is not None

else f"{self.DOCS.\_NONE\_}"

)

args\_format = (

", ".join([f"{var} ({dtype})" for var, dtype in cache.fct\_args.items()])

if cache.fct\_args is not None

else f"{self.DOCS.\_NONE\_}"

)

fct\_state\_lst.append(

f"{cache.type\_name} {cache.fct\_name}({args if args != "" else f"{self.DOCS.\_NONE\_}"}): {self.DOCS.\_FUNCTION\_NAME\_}: "

f"{cache.fct\_name}, {self.DOCS.\_ARGUMENT\_}: {args\_format if args\_format != "" else f"{self.DOCS.\_NONE\_}"}"

)

if fct\_state\_lst == []:

fct\_state\_lst.append(f"{self.DOCS.\_NONE\_}")

return fct\_state\_lst

def \_preprocessing\_before\_output(self) -> List[str]:

walkthroughs = self.parser.merged\_walkthrough\_lst

conditions = self.parser.merged\_conditions\_lst

relatives = self.parser.relatives\_lst

outputs = []

outputs.append(f"{self.DOCS.\_FUNCTION\_STATES\_}: ")

fct\_state\_lst = self.\_format\_fct\_statements()

outputs.extend(fct\_state\_lst)

outputs.append("")

outputs.append(f"{self.DOCS.\_FUNCTION\_REFS\_}: ")

for cond, call\_path in zip(conditions, walkthroughs):

if cond == "":

outputs.append(call\_path)

continue

cond = self.\_format\_cond(cond)

outputs.append(f"{self.DOCS.\_WHILE\_} {cond} {self.DOCS.\_END\_WHILE\_}: {call\_path}")

outputs.append("")

outputs.append(f"{self.DOCS.\_ARGUMENT\_RELAS\_}: ")

if relatives == []:

outputs.append(f"{self.DOCS.\_NONE\_}")

else:

for rela in relatives:

outputs.append(f"{rela}")

return outputs

def \_format\_cond(self, cond: str) -> str:

cond\_lst: List[str] = cond.split(" and ")

for idx in range(len(cond\_lst)):

if "not" in cond\_lst[idx]:

logger.debug(f"prev cond batch: {cond\_lst[idx]}")

cond\_lst[idx] = cond\_lst[idx].replace("not ", "")

if "<" in cond\_lst[idx] or "<=" in cond\_lst[idx]:

cond\_lst[idx] = cond\_lst[idx].replace("<", ">=").replace("<=", ">")

logger.debug(f"post cond batch: {cond\_lst[idx]}")

continue

if ">" in cond\_lst[idx] or ">=" in cond\_lst[idx]:

cond\_lst[idx] = cond\_lst[idx].replace(">", "<=").replace(">=", "<")

logger.debug(f"post cond batch: {cond\_lst[idx]}")

continue

if "==" in cond\_lst[idx]:

cond\_lst[idx] = cond\_lst[idx].replace("==", "!=")

else:

cond\_lst[idx] = cond\_lst[idx].replace("!=", "==")

logger.debug(f"post cond batch: {cond\_lst[idx]}")

return f" {self.DOCS.\_AND\_} ".join(cond\_lst)

（4）UI：

class GUI:

def \_\_init\_\_(self) -> None:

self.root = tk.Tk()

self.root.title("Petit Tool")

self.frame = tk.Frame(self.root)

self.frame.pack(pady=10)

# default language

self.docs = DOCS\_EN

# input box

self.inputs = scrolledtext.ScrolledText(self.frame, wrap=tk.WORD, width=80, height=20, font=("Arial", 12))

self.inputs.grid(row=0, column=0, columnspan=4, padx=10, pady=5)

# NOTE: output config deprecated, all set to True as default

# output config check box

self.output\_fct\_cache = tk.BooleanVar(value=True)

self.output\_walkthroughs = tk.BooleanVar(value=True)

self.output\_relatives = tk.BooleanVar(value=True)

# tk.Checkbutton(frame, text="Output Function States", variable=output\_fct\_cache).grid(

# row=1, column=2, padx=5, pady=5

# )

# tk.Checkbutton(frame, text="Output Function Reference", variable=output\_walkthroughs).grid(

# row=1, column=3, padx=5, pady=5

# )

# tk.Checkbutton(frame, text="Output Argument Relations", variable=output\_relatives).grid(

# row=1, column=4, padx=5, pady=5

# )

# output box

self.outputs = scrolledtext.ScrolledText(

self.frame, wrap=tk.WORD, width=80, height=15, font=("Arial", 12), state="disabled"

)

self.outputs.grid(row=3, column=0, columnspan=4, padx=10, pady=5)

# file loading button

self.load\_button = tk.Button(self.frame, text=self.docs.\_LOAD\_FILE\_, command=lambda: tk\_load\_cfile(self.inputs))

self.load\_button.grid(row=1, column=0)

self.parse\_button = tk.Button(

self.frame,

text="Parse Code",

command=lambda: Core(

DOCS=self.docs,

src\_code=None,

tk\_inputs=self.inputs,

tk\_outputs=self.outputs,

output\_fct\_cache=self.output\_fct\_cache,

output\_relatives=self.output\_relatives,

output\_walkthroughs=self.output\_walkthroughs,

),

)

self.parse\_button.grid(row=1, column=1)

self.switch\_button = tk.Button(self.frame, text=self.docs.\_SWITCHER\_, command=self.switcher)

self.switch\_button.grid(row=1, column=3, pady=10)

self.root.mainloop()

def switcher(self) -> None:

self.docs = DOCS\_CN if self.docs == DOCS\_EN else DOCS\_EN

# renew UI text

self.load\_button.config(text=self.docs.\_LOAD\_FILE\_)

self.parse\_button.config(text=self.docs.\_PARSE\_CODE\_)

self.switch\_button.config(text=self.docs.\_SWITCHER\_)

**五、测试数据及其结果分析**

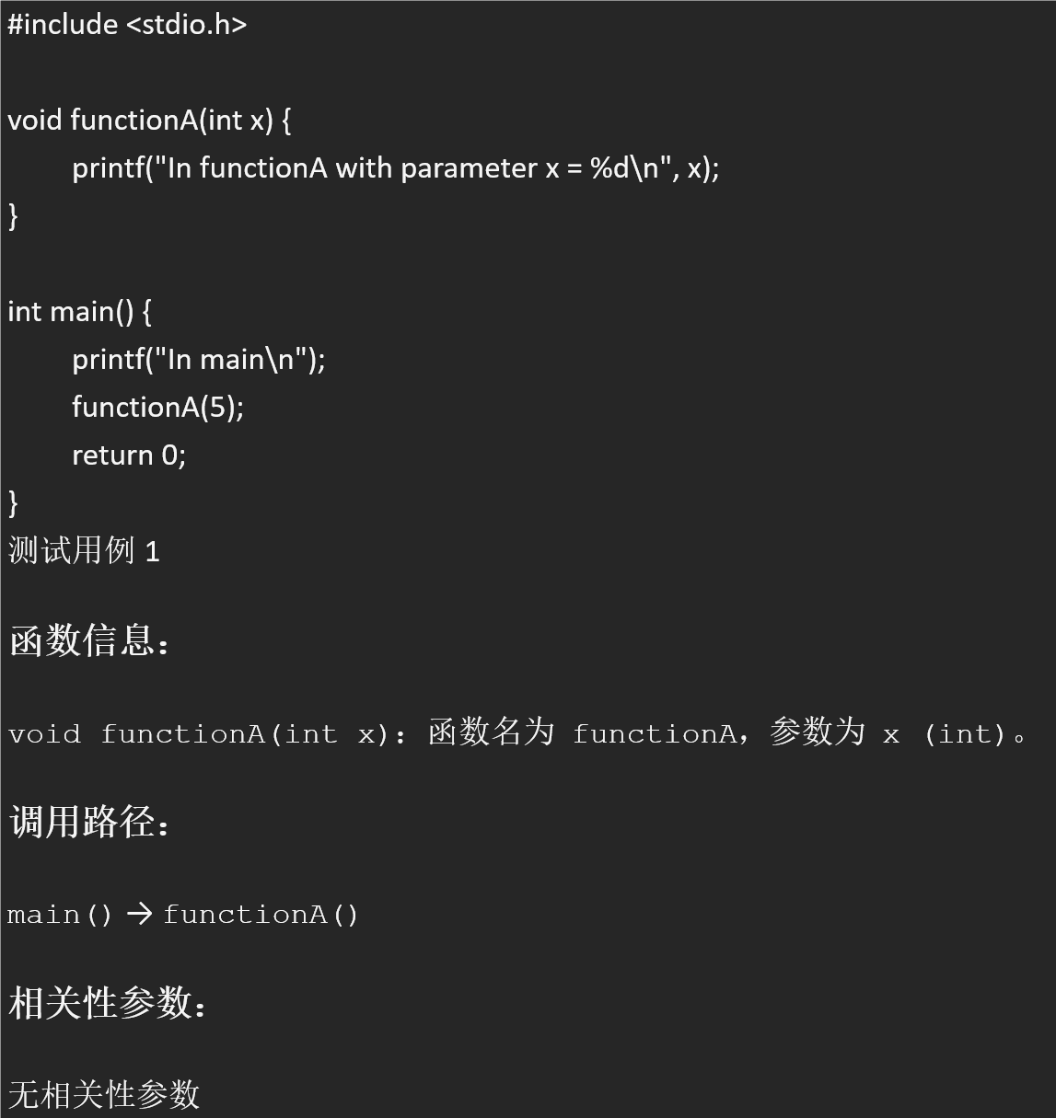


图4测试一

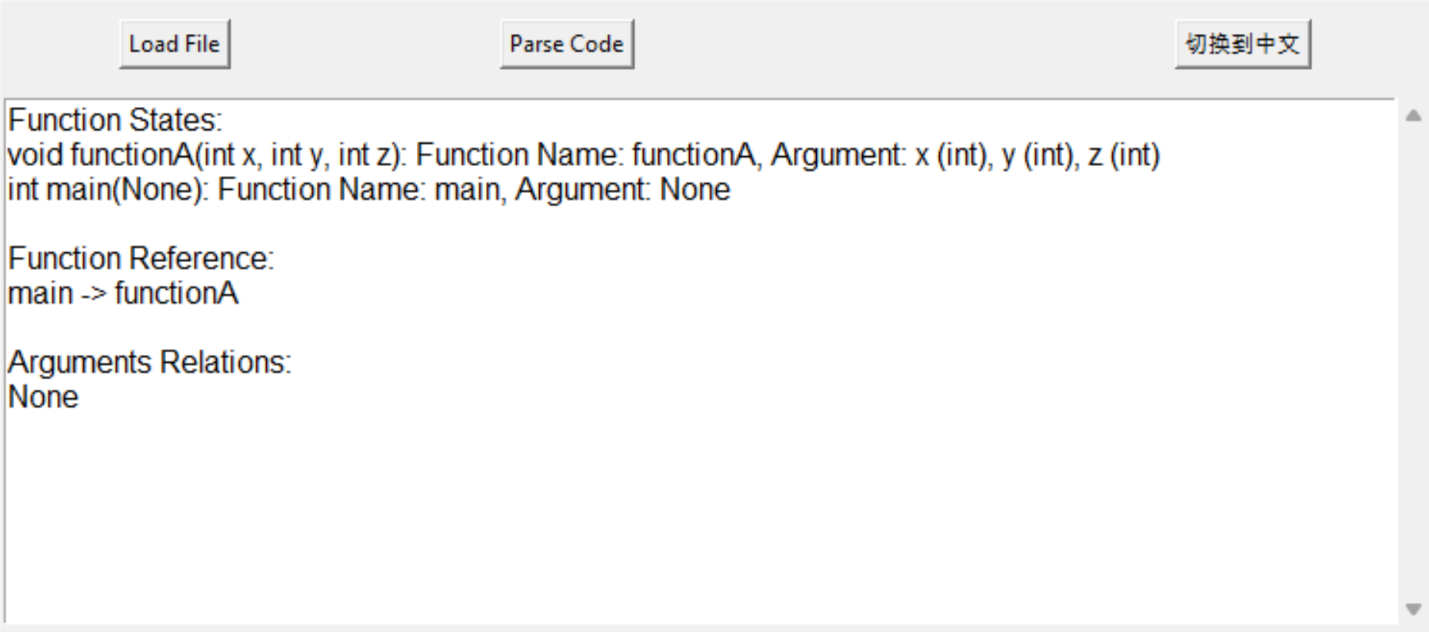


图5结果1

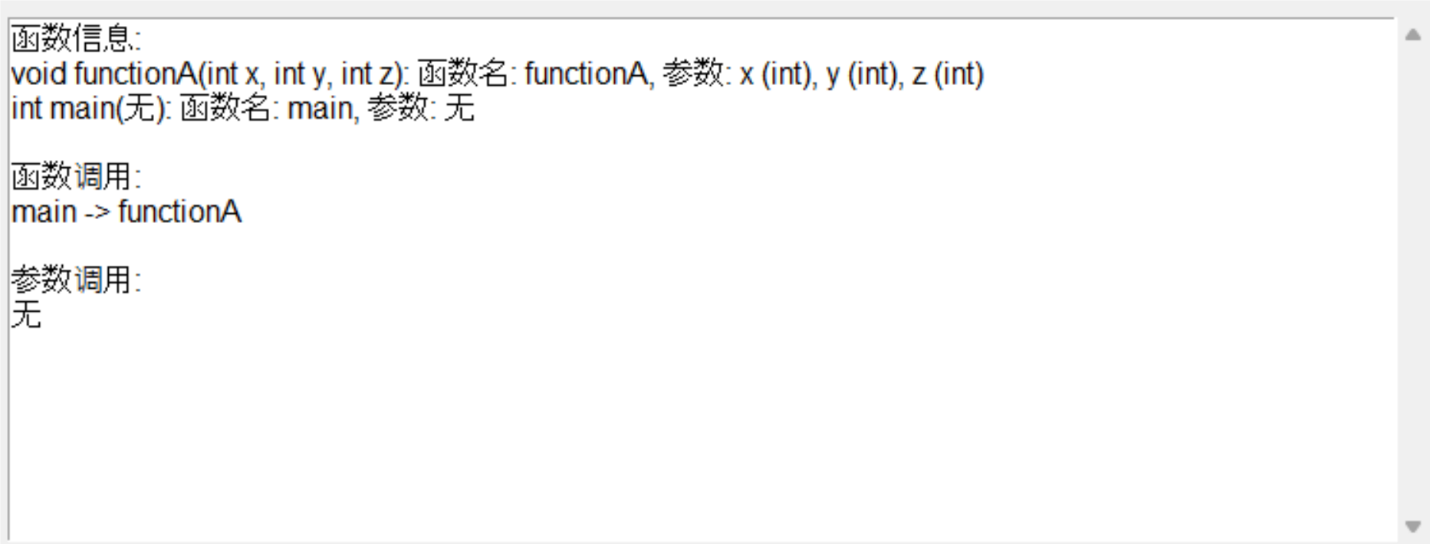


图6结果2



图7测试二

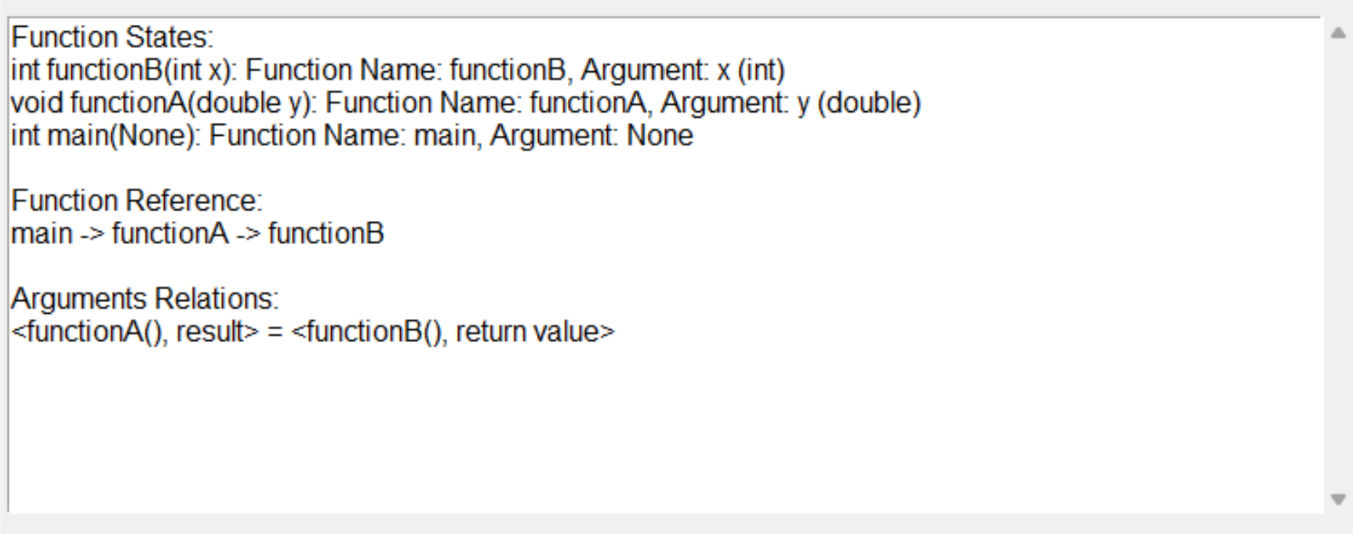


图8结果1

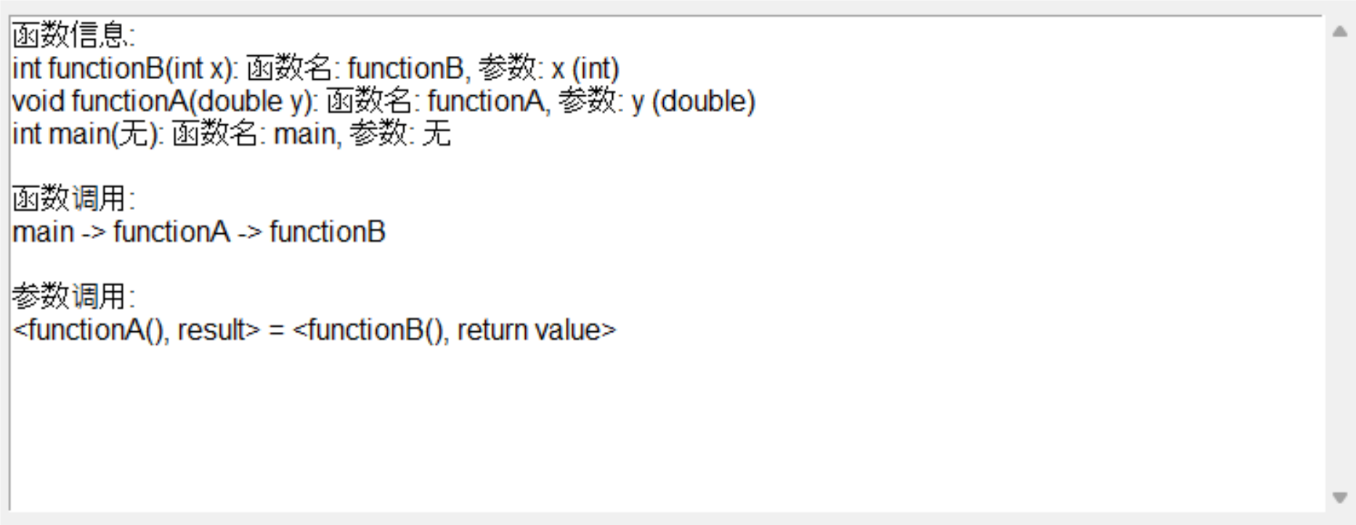
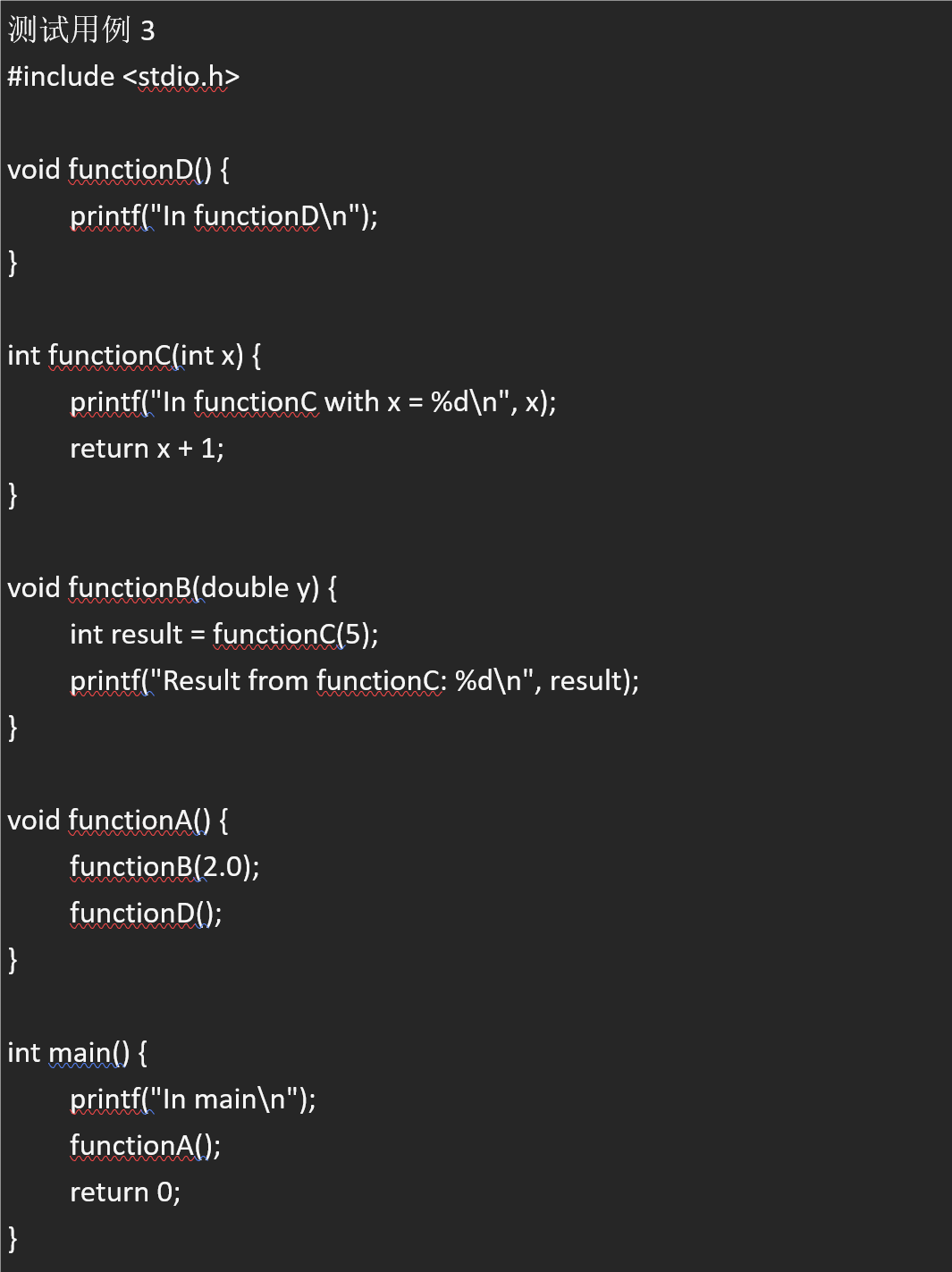


图9结果2



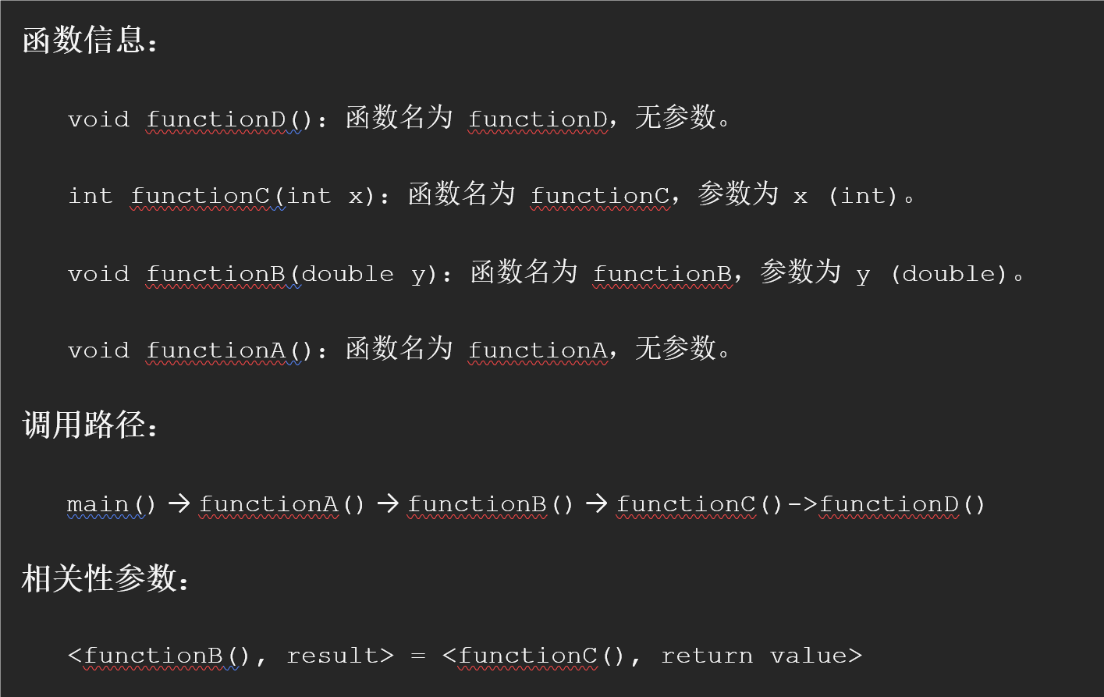


图10测试三

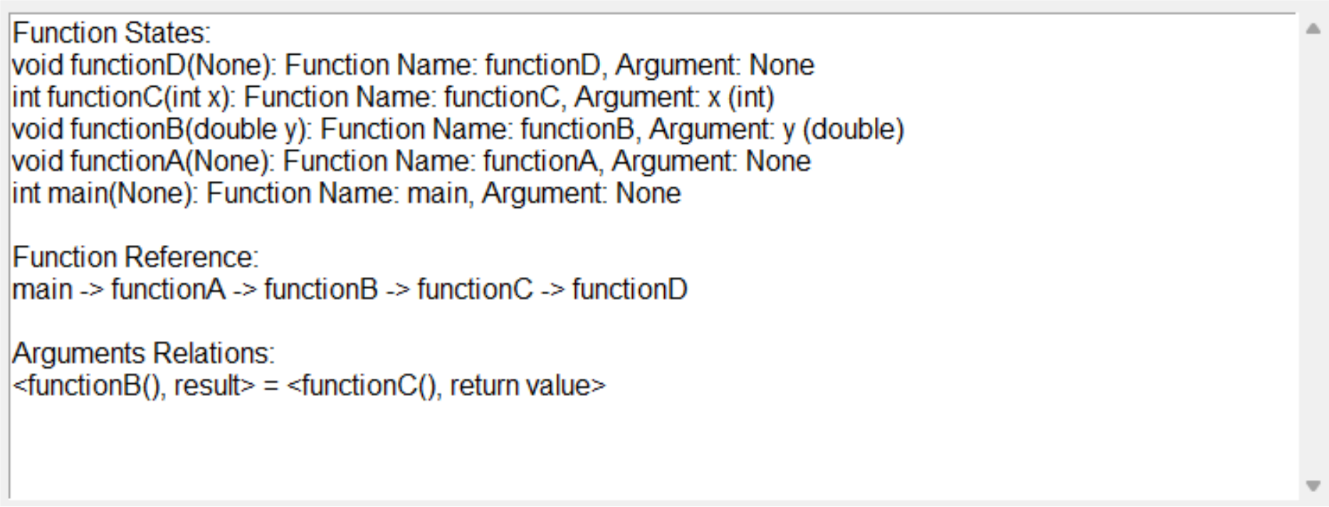
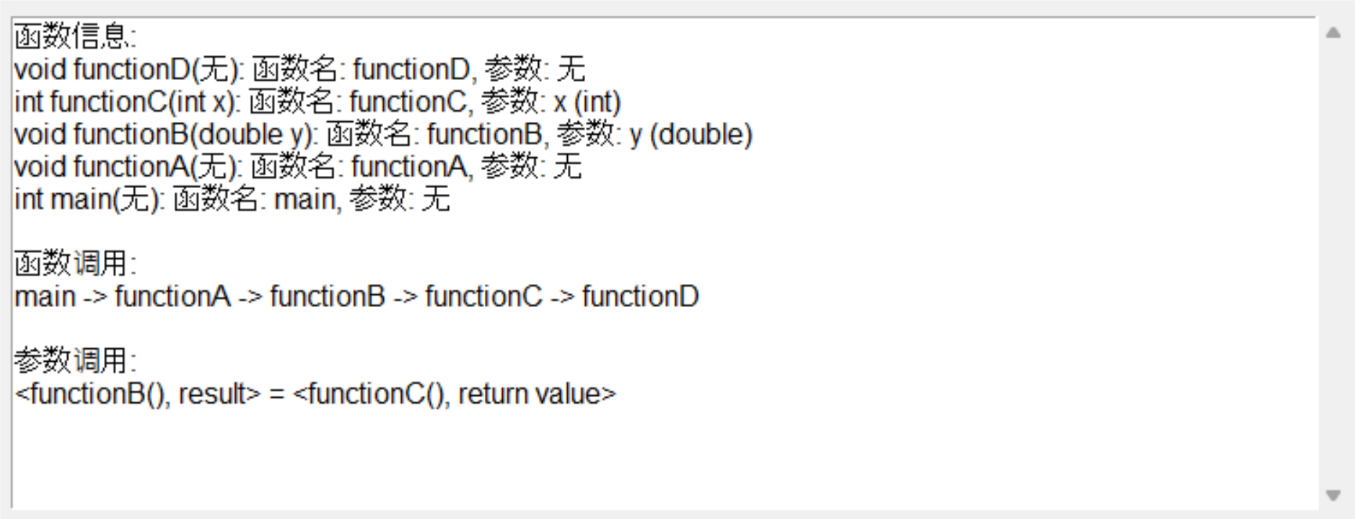
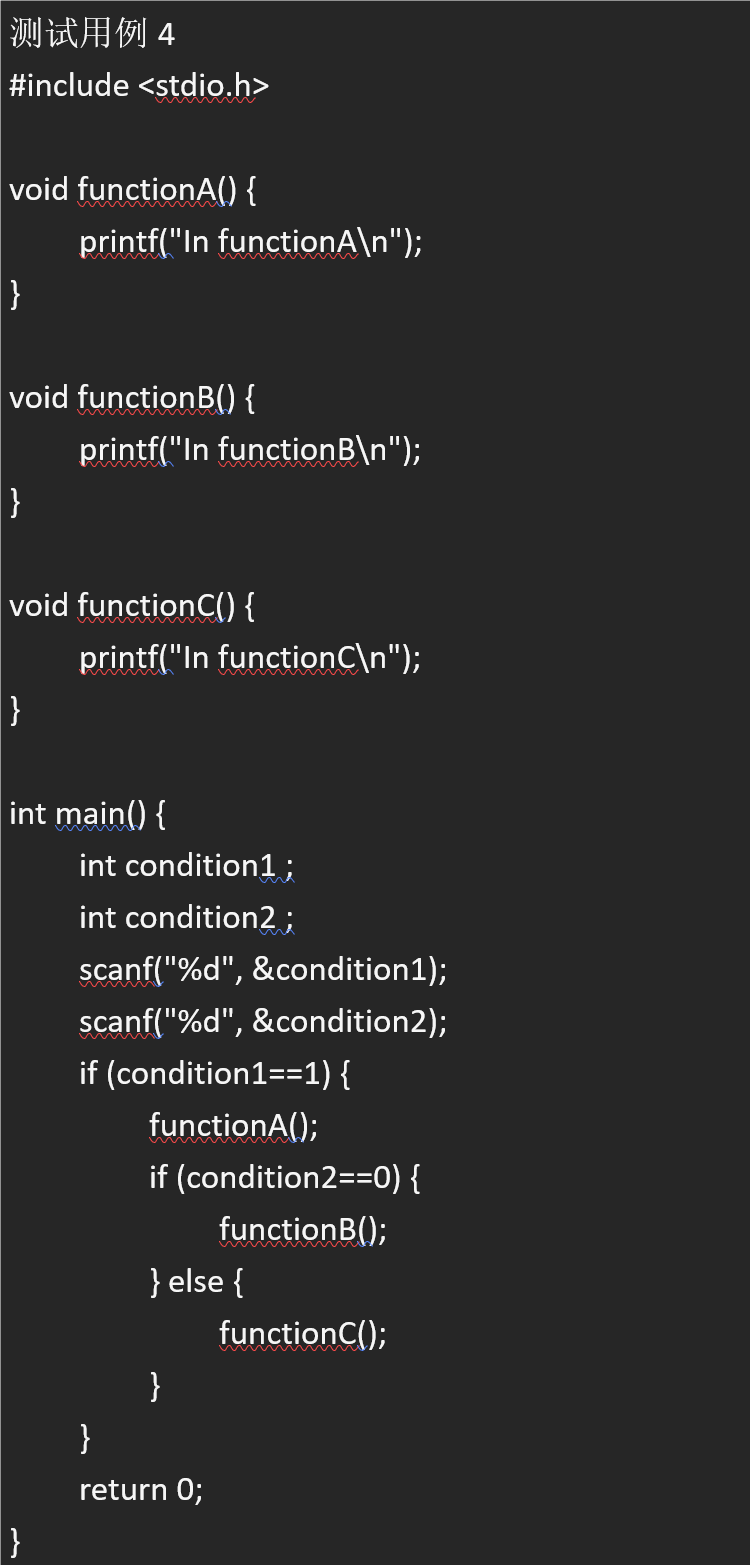


图11结果1

图12结果2



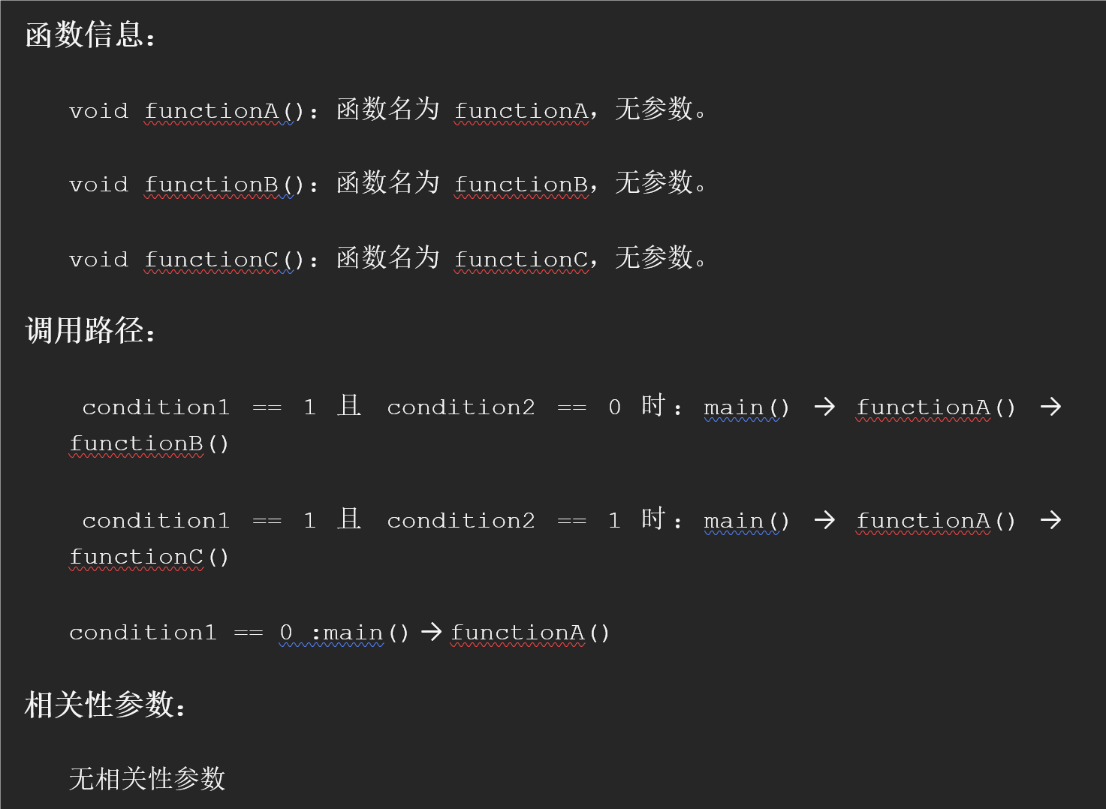


图13测试四

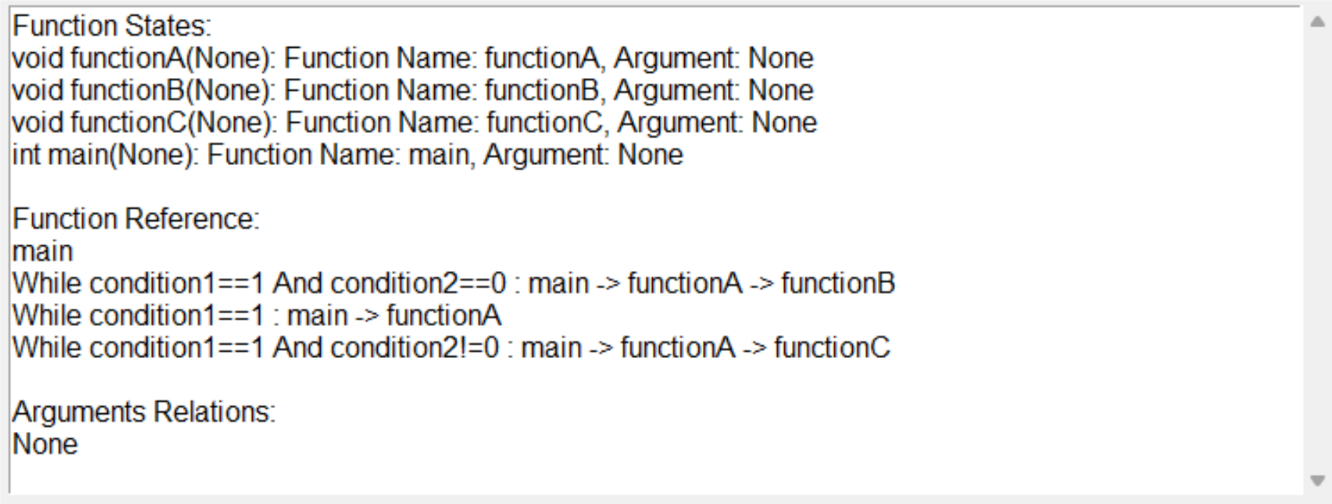


图15结果1

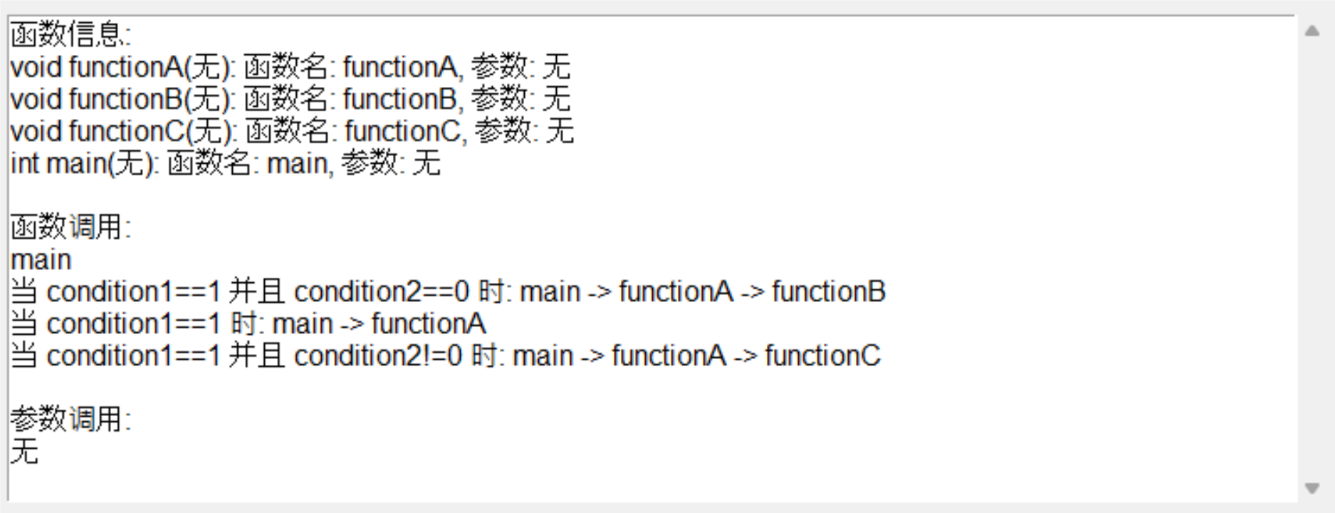
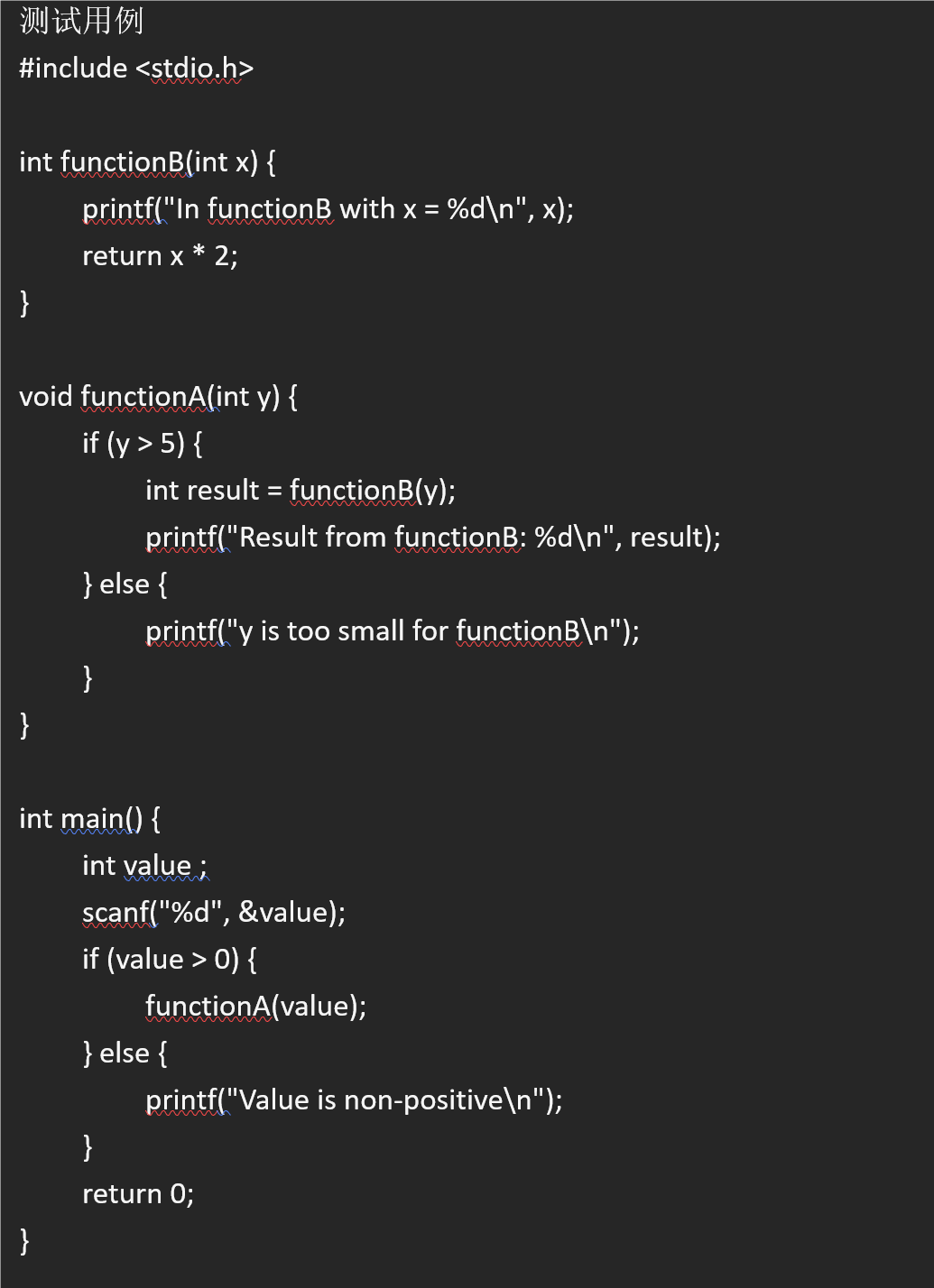


图16结果2



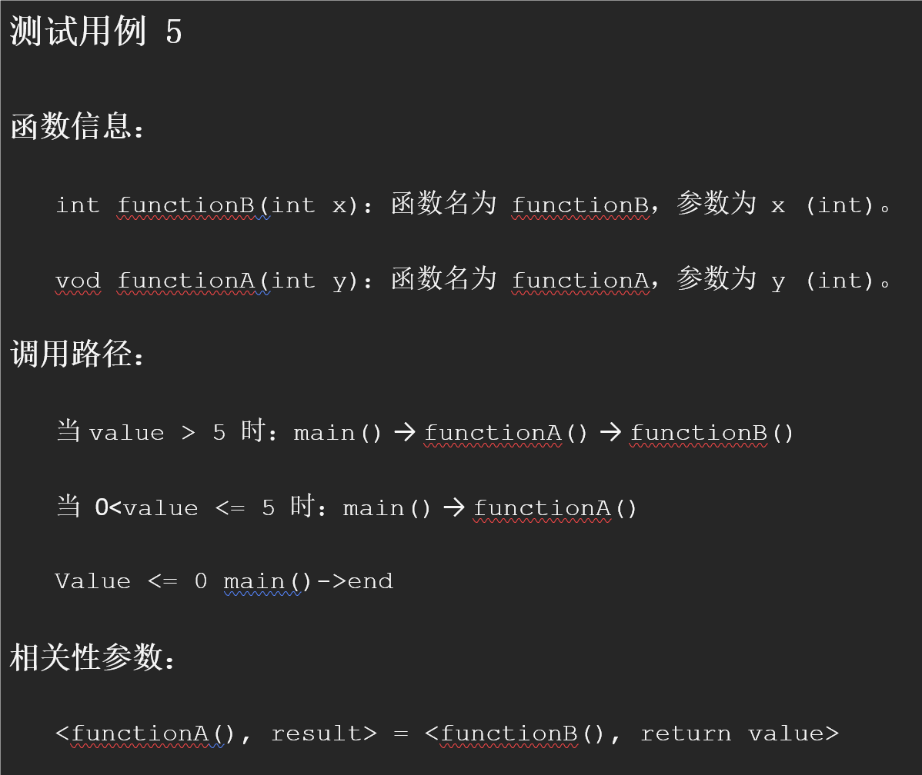


图17测试五

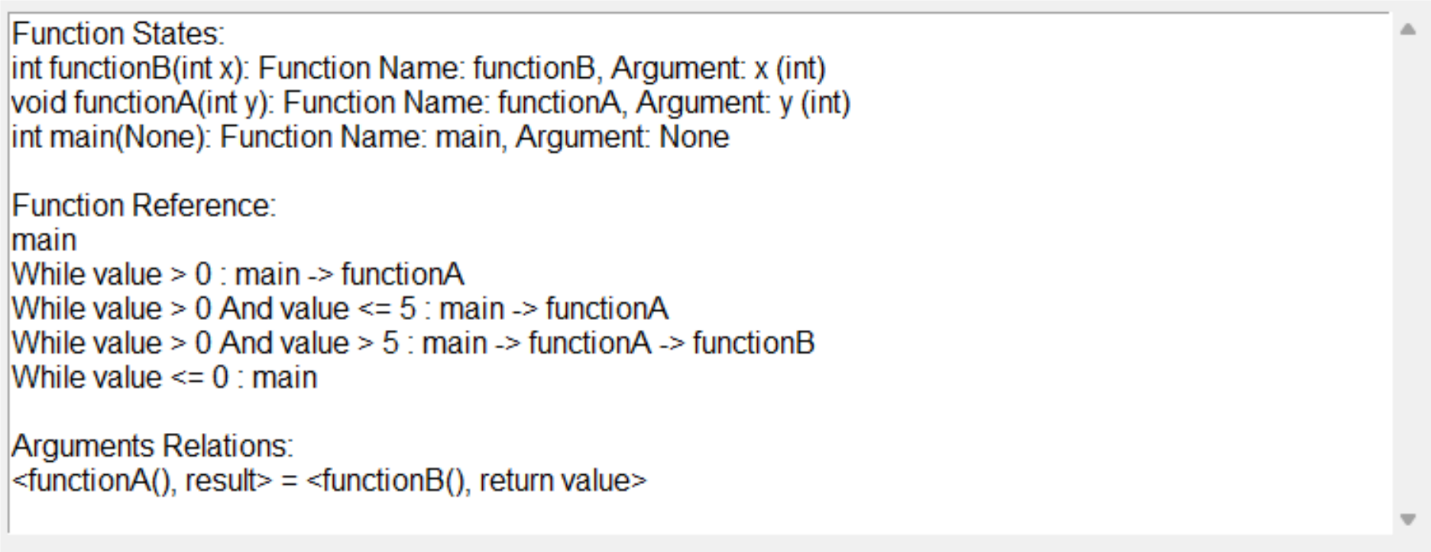


图18结果1

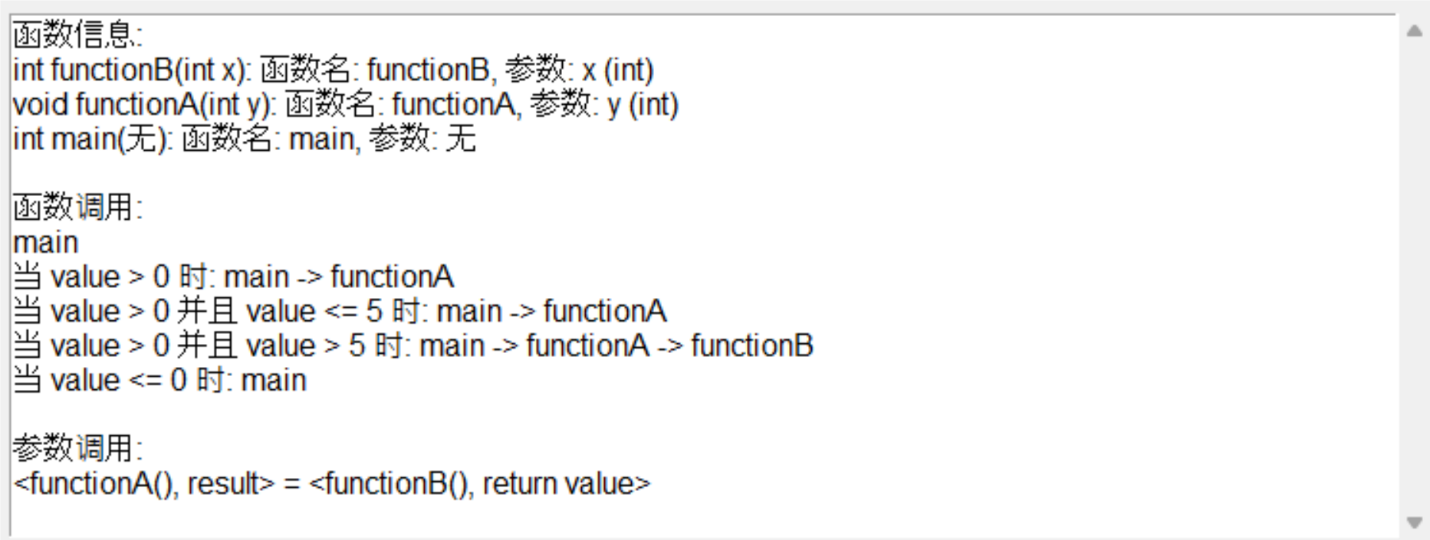


图18结果2

**六、算法设计和程序调试过程中的问题**

**问题1**：难以维护和存储函数信息

**解决方法**：用python自带函数库data class来存储，再写一个维护的类来处理，输入输出和调用。

**问题2**：函数调用、迭代难以处理

**解决方法**：利用动态规划的逻辑，模拟C语言函数的迭代。

**问题3**：if else 条件判断语句

**解决方法**：将if和else视为一个新的函数，其内容为函数内容，用问题2的处理方法同样进行迭代，并在其状态转移方程中加入condition作为执行条件，这在最后将被输出。

**问题4**：冗余输出

**解决方法**：如图18，“value > 0 并且 value > 5”该输出完全可以由“value > 5”单独完成，但由于问题3的解决方案导致这一点只能在后续的处理中完成。然而在此由于认为其对结果的影响不大，并没有采取这个措施，保留了原问题。

**七、课程设计总结**

基于问题4，可以进一步优化输入输出结果。另外，该系统基于正则表达式，对内容的提取泛化程度极其有限，而却又基于对C语言内容和逻辑的模拟，于是导致该系统的容错率极低。对此，我认为还是应当回归基本的字符串处理，这样可以更好地模拟C的编译模式。C是一个对格式要求较高的语言，而这也恰恰使字符串处理的准确率、容错率和效率大大提升。另一方面，正则表达式的可读性很低，运行效率也很低。利用字符串处理，或许能提高程序的可读性、可拓展性，在一定程度上降低编译和维护的难度。

**其他要求：**

1、最后阶段请认真完成本次程序设计报告的电子文档，报告内容不得少于15页。

2、每个学生最后提交的材料：

① 程序设计报告的电子文档(起名为：**算法与数据结构程序设计报告\_学号.doc)**，单独作为一个文件提交，所有人收齐后请班长给助教；

② 压缩文件一个，起名为：**算法与数据结构程序设计\_学号.rar(如：算法与数据结构程序设计\_B16040601.rar)。**该压缩文件中包含下列几个文件：程序设计报告的电子文档(起名为：**算法与数据结构程序设计报告\_学号.doc)**、源程序文件(起名为：**题目名\_学号.cpp，如：题目1\_ B16040601.cpp)、**可能用到的原始数据文件(如：**stud.dat**)、编译链接后得到的可执行文件(起名为：**题目1\_学号.exe)**

③ 每位同学请打印纸质版交给班长，**注意封面是正反两面打印在一页上**。

3、由班长建立根文件夹——算法与数据结构程序设计\_B160406；其中包含若干个子文件夹，每个子文件夹的名字分别为同学们所选择的不同组合的题目，例如“**题目1**”或者“**题目4**”等；在这若干个子文件夹中，例如“**题目1**”，应该包含完成该相同组合题目的所有同学(例如B16040601 B16040606……)的相关文件，即把第2点得到的压缩文件“**算法与数据结构程序设计\_学号.rar**”放在其中即可。完成后班长压缩该文件夹给助教。