ASSIGNMENT 4:
NORTHWESTERN MEMORIAL HEALTHCARE AGENT

Kevin Geidel
MSDS 442: AI Agent Design & Development
Northwestern University

May 25, 2025

Requirements 1 and 2: Graph the agent with LangChain/LangGraph

The construction of the agent and accompanying graph begins with the creation of the elements (classes and functions) that serve as nodes and edges in our graph. This begins as early as cell 2 (see appendix) with the creation of the `InquiryState` class that tracks the current values and message history for the agent cluster. The actual assembly of the graph itself occurs in cell 13. However the components are instantiated above that. Following the logic in cell 13 we first instantiate an empty graph:

```
builder = StateGraph(InquiryState)
```

The `builder` object has `add_node` and `add_edge` methods that allow us to assemble the components created in cells 2-12. We use `add_conditional_edges` to inform the graph that the next node will be determined dynamically, depending on agent/user interactions. The output is displayed graphically in cell 14 (reproduced in figure 1 below.)

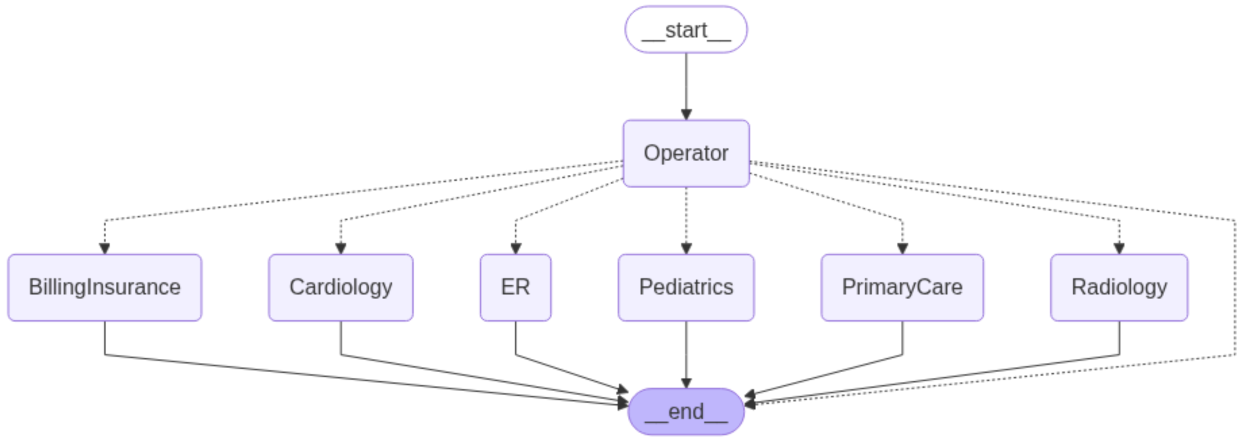


Figure 1: Graphical depiction of the Northwestern Memorial Healthcare Agent.

We can compare the workflow in figure 1 to the flowchart of desired functionality provided by Dr. Bader in the assignment requirements (figure 2.) Each desired agent is represented as a node in the `LangGraph`. Edges represent the state being transferred from one agent to the next.

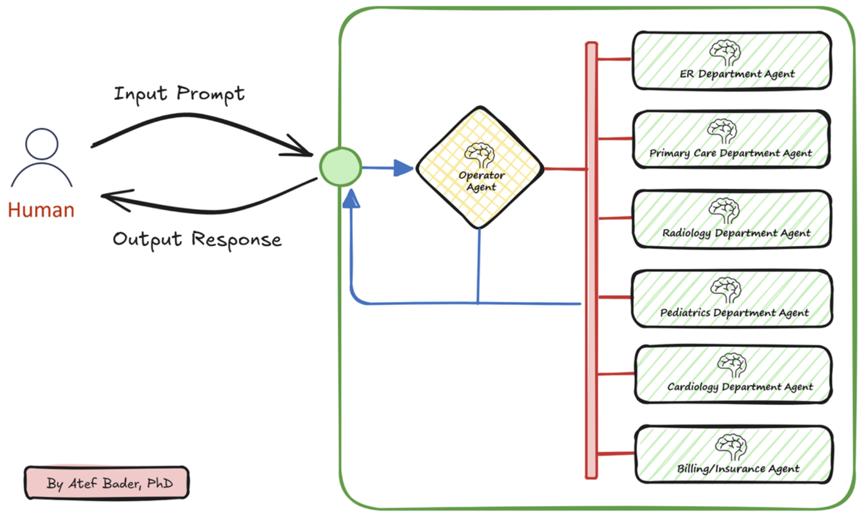
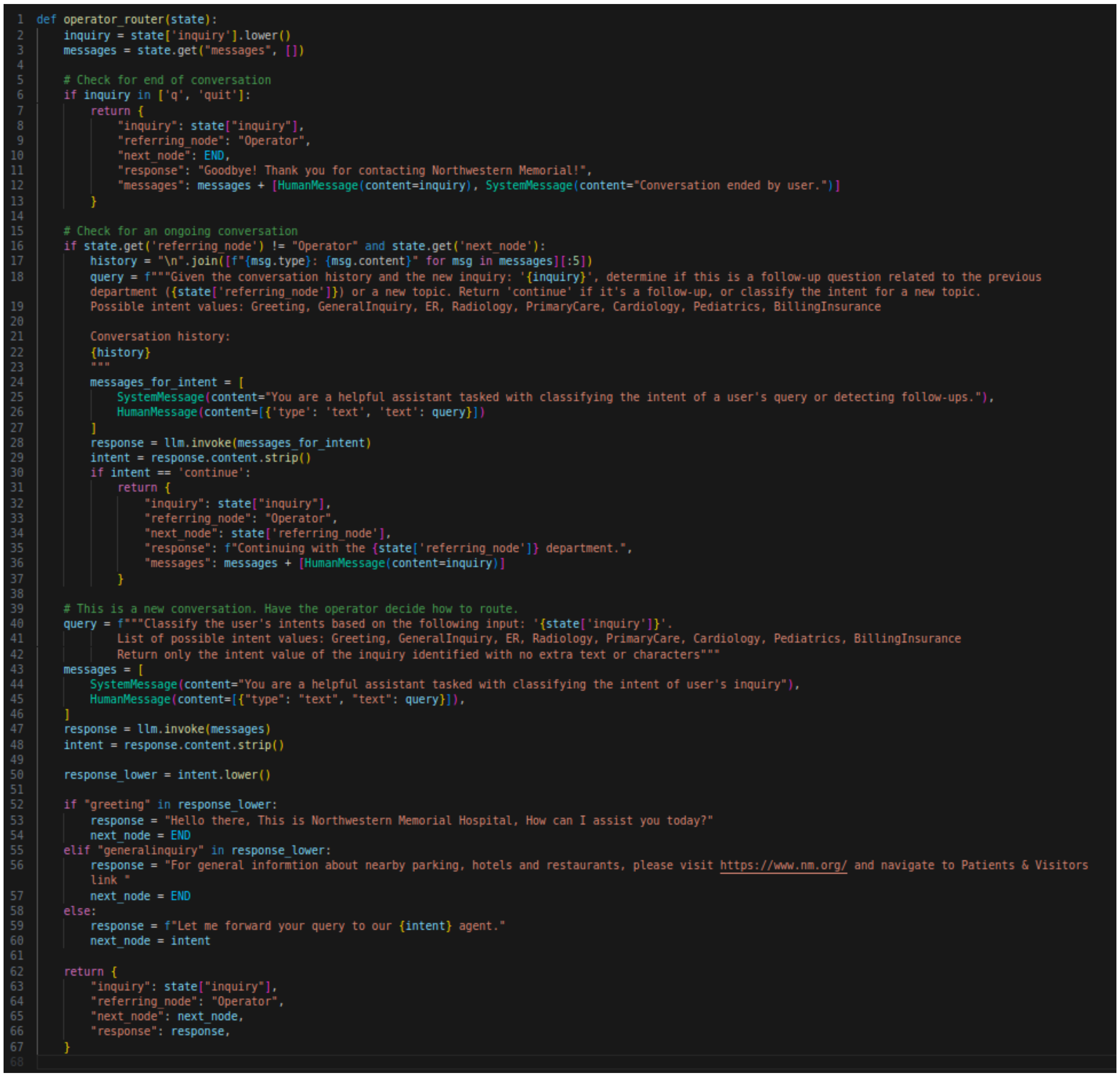

Input Prompt
Output Response
Human
Operator Agent
ER Department Agent
Primary Care Department Agent
Radiology Department Agent
Pediatrics Department Agent
Cardiology Department Agent
Billing/Insurance Agent
By Atef Bader, PhD

Figure 2: The desired workflow as provided by the assignment objectives.

Requirement 3: The operator agent

An operator agent acts as a router to direct queries to and from the various department agents. The operator is constructed in cell 5 (reproduced in figure 3 with line numbers.) Lines 2 and 3 extract the current inquiry and message history respectively. Lines 5-13 check for any termination conditions and end the conversation gracefully if present.

In order to accommodate multi-turn conversations the operator must be able to detect and respond accordingly to existing chat history. The conditional in line 16, `if state.get('referring_node') != "Operator" and state.get('next_node')`, will evaluate to True if the department agents have already interacted with the user at least once (`next_node` is not set until initially.) The five most recent messages are extracted from the state in line 17 (limit of five to avoid token overflows.) In lines 18-29 the operator invokes the LLM with a prompt crafted to evaluate the chat history and determine if the latest user query is part of an on going conversation (i.e. a follow-up question, an additional question or providing requested information.) If the LLM determines this is the case the inquiry is forwarded back to the referring node/department. This query can also decide that the latest inquiry is best served by a different department and 'transfer' the inquiry to it (instead of the referring node.)



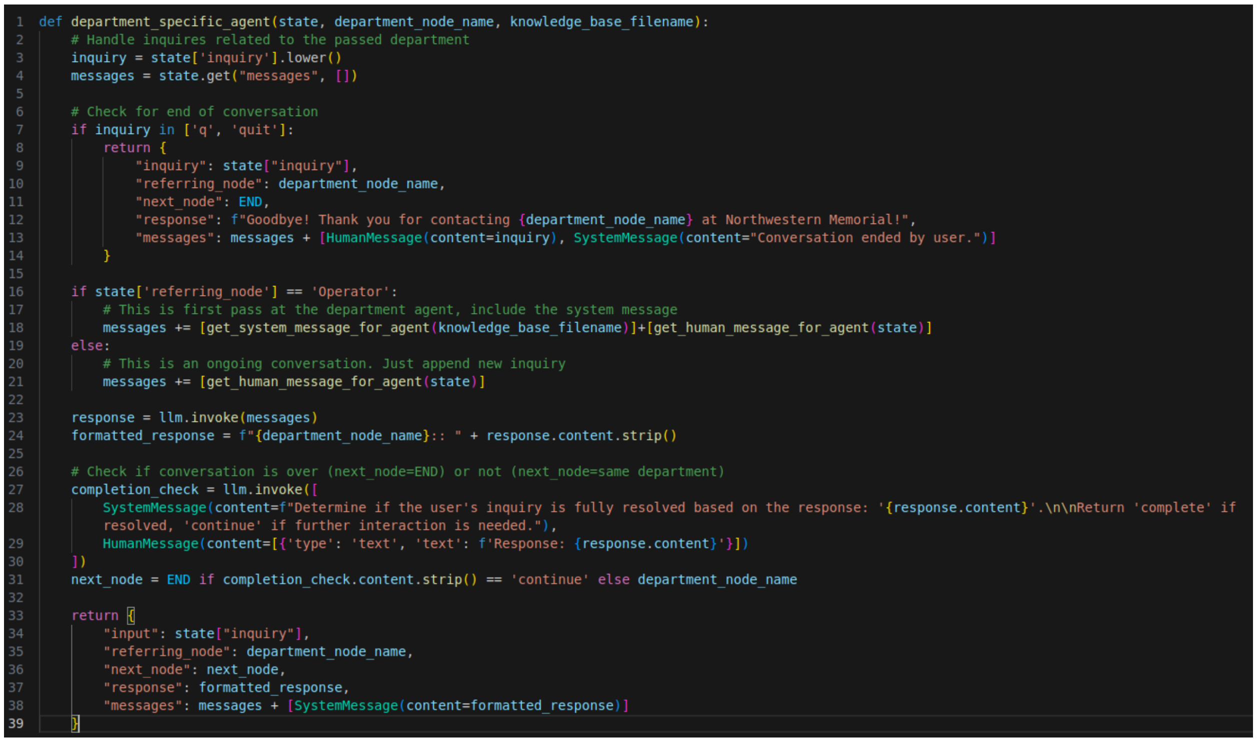
```
1  def operator_router(state):
2      inquiry = state['inquiry'].lower()
3      messages = state.get("messages", [])
4  
5      # Check for end of conversation
6      if inquiry in ['q', 'quit']:
7          return {
8              "inquiry": state["inquiry"],
9              "referring_node": "Operator",
10             "next_node": END,
11             "response": "Goodbye! Thank you for contacting Northwestern Memorial!",
12             "messages": messages + [HumanMessage(content=inquiry), SystemMessage(content="Conversation ended by user.")]
13         }
14 
15     # Check for an ongoing conversation
16     if state.get('referring_node') != "Operator" and state.get('next_node'):
17         history = "\n".join([f"{msg.type}: {msg.content}" for msg in messages][:5])
18         query = f"""Given the conversation history and the new inquiry: '{inquiry}', determine if this is a follow-up question related to the previous
           department ({state['referring_node']}) or a new topic. Return 'continue' if it's a follow-up, or classify the intent for a new topic.
19         Possible intent values: Greeting, GeneralInquiry, ER, Radiology, PrimaryCare, Cardiology, Pediatrics, BillingInsurance
20 
21         Conversation history:
22         {history}
23         """
24         messages_for_intent = [
25             SystemMessage(content="You are a helpful assistant tasked with classifying the intent of a user's query or detecting follow-ups."),
26             HumanMessage(content=[{'type': 'text', 'text': query}])
27         ]
28         response = llm.invoke(messages_for_intent)
29         intent = response.content.strip()
30         if intent == 'continue':
31             return {
32                 "inquiry": state["inquiry"],
33                 "referring_node": "Operator",
34                 "next_node": state['referring_node'],
35                 "response": f"Continuing with the {state['referring_node']} department.",
36                 "messages": messages + [HumanMessage(content=inquiry)]
37             }
38 
39     # This is a new conversation. Have the operator decide how to route.
40     query = f"""Classify the user's intents based on the following input: '{state['inquiry']}'.
41             List of possible intent values: Greeting, GeneralInquiry, ER, Radiology, PrimaryCare, Cardiology, Pediatrics, BillingInsurance
42             Return only the intent value of the inquiry identified with no extra text or characters"""
43     messages = [
44         SystemMessage(content="You are a helpful assistant tasked with classifying the intent of user's inquiry"),
45         HumanMessage(content=[{"type": "text", "text": query}]),
46     ]
47     response = llm.invoke(messages)
48     intent = response.content.strip()
49 
50     response_lower = intent.lower()
51 
52     if "greeting" in response_lower:
53         response = "Hello there, This is Northwestern Memorial Hospital, How can I assist you today?"
54         next_node = END
55     elif "generalinquiry" in response_lower:
56         response = "For general informtion about nearby parking, hotels and restaurants, please visit https://www.nm.org/ and navigate to Patients & Visitors
           link "
57         next_node = END
58     else:
59         response = f"Let me forward your query to our {intent} agent."
60         next_node = intent
61 
62     return {
63         "inquiry": state["inquiry"],
64         "referring_node": "Operator",
65         "next_node": next_node,
66         "response": response,
67     }
68 
```

Figure 3: The operator/router is instantiated in cell 5.

By line 39 we are on the other side of the existing conversation conditional and we can be confident that the inquiry marks a new conversation. Lines 40-50 repeat a similar query to the LLM- classifying user intent for the purposes of routing the inquiry. The difference here is there prompt does not have to consider if the inquiry is related to historic messages at all. Lines 52-57 set some hard coded responses and routes for some specific intents but, in lines 62-67, the inquiry is routed to the appropriate department specific agent.

Requirement 4: Department specific agents

Cells 7-12 instantiate functions that represent the six agents and their respective departments. The reader will likely note these six functions simply return the output of a common function, passed department specific arguments. The function, `department_specific_agent`, is an abstracted agent that performs the same steps for each department agent. These steps include loading conversation history, checking for (and reacting to) termination conditions, determining if this is an ongoing or new conversation, loading the department's knowledge base into context for new conversations, checking if the user's issue has been resolved and forwarding the state to the appropriate next node. The base department agent is defined in cell 6 and reproduced, with line numbers, in figure 4.



```
1  def department_specific_agent(state, department_node_name, knowledge_base_filename):
2      # Handle inquires related to the passed department
3      inquiry = state['inquiry'].lower()
4      messages = state.get("messages", [])
5
6      # Check for end of conversation
7      if inquiry in ['q', 'quit']:
8          return {
9              "inquiry": state["inquiry"],
10             "referring_node": department_node_name,
11             "next_node": END,
12             "response": f"Goodbye! Thank you for contacting {department_node_name} at Northwestern Memorial!",
13             "messages": messages + [HumanMessage(content=inquiry), SystemMessage(content="Conversation ended by user.")]
14         }
15
16     if state['referring_node'] == 'Operator':
17         # This is first pass at the department agent, include the system message
18         messages += [get_system_message_for_agent(knowledge_base_filename)]+[get_human_message_for_agent(state)]
19     else:
20         # This is an ongoing conversation. Just append new inquiry
21         messages += [get_human_message_for_agent(state)]
22
23     response = llm.invoke(messages)
24     formatted_response = f"{department_node_name}:: " + response.content.strip()
25
26     # Check if conversation is over (next_node=END) or not (next_node=same department)
27     completion_check = llm.invoke([
28         SystemMessage(content=f"Determine if the user's inquiry is fully resolved based on the response: '{response.content}'.\n\nReturn 'complete' if
           resolved, 'continue' if further interaction is needed."),
29         HumanMessage(content=[{'type': 'text', 'text': f'Response: {response.content}'}])
30     ])
31     next_node = END if completion_check.content.strip() == 'continue' else department_node_name
32
33     return {
34         "input": state["inquiry"],
35         "referring_node": department_node_name,
36         "next_node": next_node,
37         "response": formatted_response,
38         "messages": messages + [SystemMessage(content=formatted_response)]
39     }
```

Figure 4: The abstracted department agent is instantiated in cell 6.

Lines 1-14 are similar to those of the operator- extracting what is needed from the current state and handling termination conditions (if detected.) This could likely be abstracted as well since the only differences related to specific hard coded messages to customize the experience (i.e. 'Thank you for contacting the Cardiology Department!' vs 'Come again soon!') Lines 16-21 load the latest (or first) user inquiry into messages ahead of invoking the LLM. The conditional is required to determine if the workflow has been through this department yet. If the referring node is the operator then this is the first pass through the department. The department specific `SystemMessage` (which is created dynamically from utility functions defined in cell 4) is injected into the conversation ahead of the `UserMessage`. If the referring node is this department this step is unnecessary and the current inquiry appended only.

The actual call to the LLM takes place in lines 23 and 24. Before the agent can properly route the inquiry to the next node it must first determine if the conversation is over (in which case `next_node` is set to `END`) or if the user's request is not resolved (and `next_node` is set back to this department.) This check is accomplished with an LLM query created in lines 27-31. The prompt is simple and asks the LLM return 'continue' if the conversation is not adequately resolved.

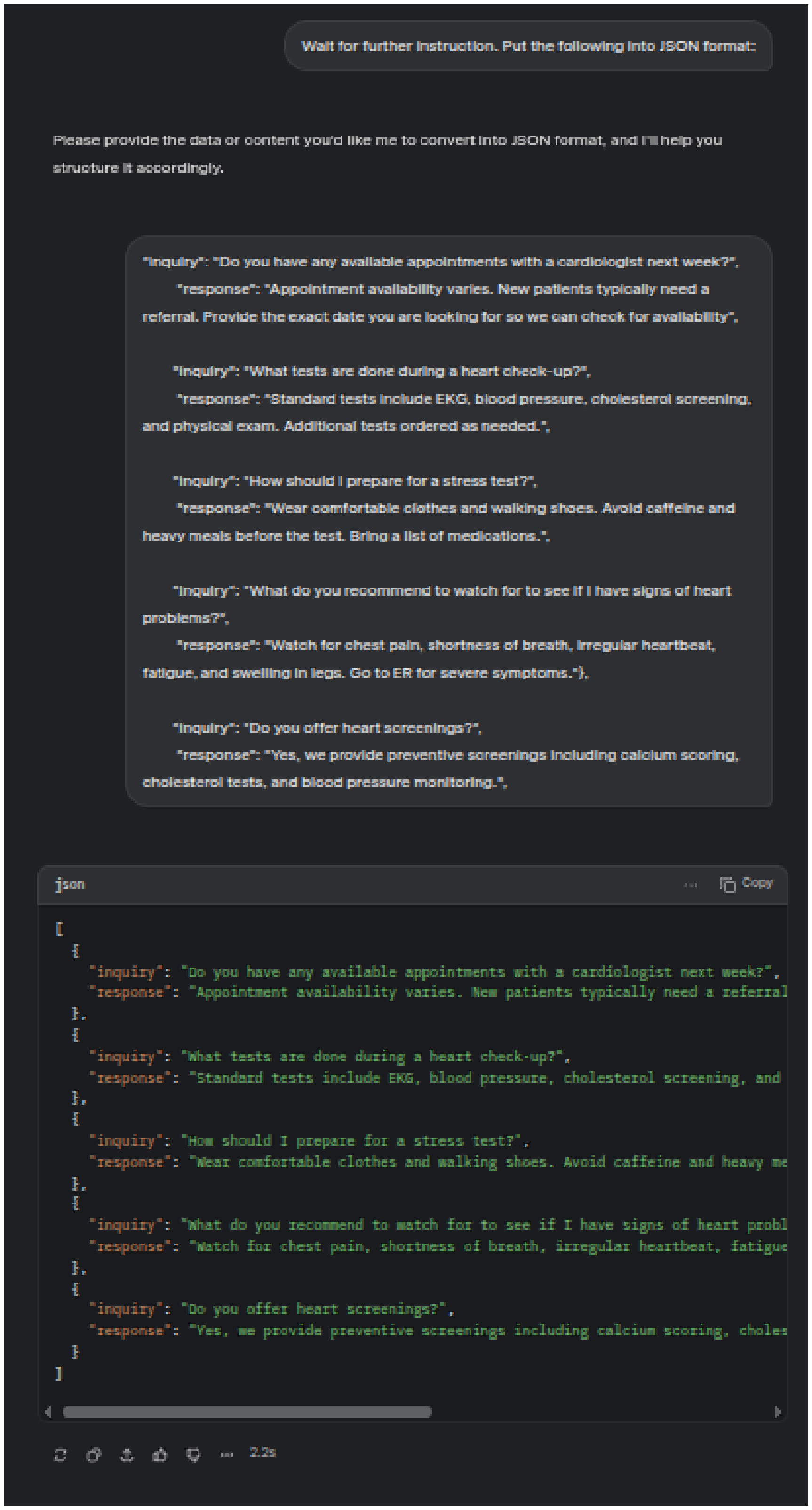
Requirement 5: Classifying intent

LLM queries are used to determine user intent in three different places. Two occur within the operator agent. One query is for classifying a new inquiry so that is can be routed to the appropriate department agent. This can be found in lines 40-48 of cell 5 (see figure 3.) The second query that the operator agent can use to infer user intent is for existing conversations. The LLM (lines 18-29 of cell 5 in figure 3) is tasked with deciding if the latest inquiry must go back to the referring node (to continue an ongoing conversation) or be routed to a new department.

The third instance of the agents using the LLM to classify intent happens in the respective departments. The LLM is used to decide if the conversation is over or needs to be routed back around for a multi-turn conversation. This is implemented in lines 26-31 of cell 6 (see figure 4.)

Requirement 6: Knowledge base data

The agents instruct the LLM to restrict answers to the information in the department's respective knowledge base. The knowledge bases consist of six JSON files (one for each department) that contain a number of question ('inquiry') and answer ('response') entries. Some of these entries were provided with the assignment. I used Grok (`https://grok.com/`) to convert the provided questions and answers into the desired JSON format (figure 5.)



Wait for further instruction. Put the following into JSON format:

Please provide the data or content you'd like me to convert into JSON format, and I'll help you structure it accordingly.

"inquiry": "Do you have any available appointments with a cardiologist next week?",
"response": "Appointment availability varies. New patients typically need a referral. Provide the exact date you are looking for so we can check for availability",

"inquiry": "What tests are done during a heart check-up?",
"response": "Standard tests include EKG, blood pressure, cholesterol screening, and physical exam. Additional tests ordered as needed.",

"inquiry": "How should I prepare for a stress test?",
"response": "Wear comfortable clothes and walking shoes. Avoid caffeine and heavy meals before the test. Bring a list of medications.",

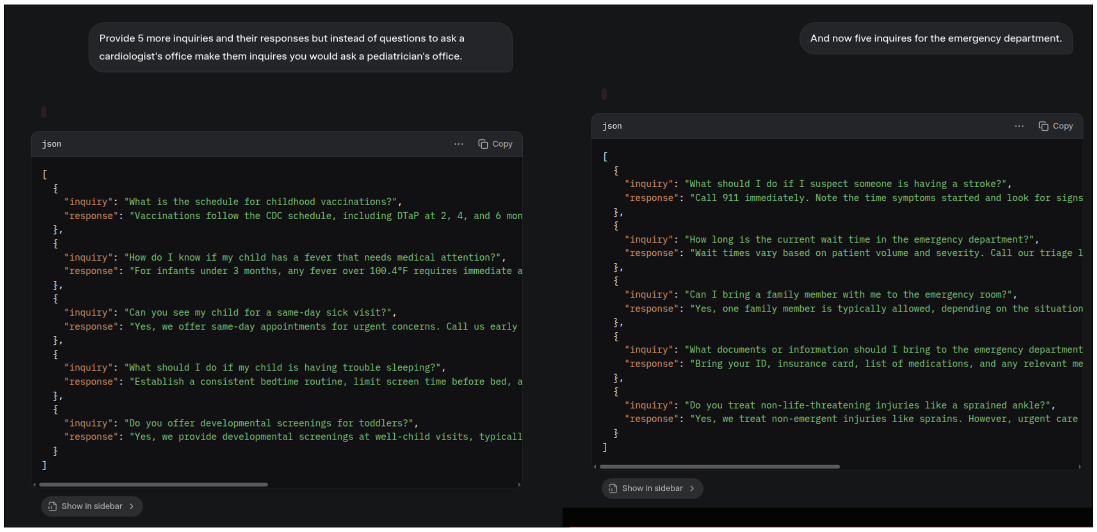
"inquiry": "What do you recommend to watch for to see if I have signs of heart problems?",
"response": "Watch for chest pain, shortness of breath, irregular heartbeat, fatigue, and swelling in legs. Go to ER for severe symptoms."},

"inquiry": "Do you offer heart screenings?",
"response": "Yes, we provide preventive screenings including calcium scoring, cholesterol tests, and blood pressure monitoring.",

```json
[
  {
    "inquiry": "Do you have any available appointments with a cardiologist next week?",
    "response": "Appointment availability varies. New patients typically need a referral
  },
  {
    "inquiry": "What tests are done during a heart check-up?",
    "response": "Standard tests include EKG, blood pressure, cholesterol screening, and
  },
  {
    "inquiry": "How should I prepare for a stress test?",
    "response": "Wear comfortable clothes and walking shoes. Avoid caffeine and heavy me
  },
  {
    "inquiry": "What do you recommend to watch for to see if I have signs of heart probl
    "response": "Watch for chest pain, shortness of breath, irregular heartbeat, fatigue
  },
  {
    "inquiry": "Do you offer heart screenings?",
    "response": "Yes, we provide preventive screenings including calcium scoring, choles
  }
]
```

Figure 5: The commercial AI, Grok, converting knowledge base entries into a JSON object.

Grok was also used to generate new questions. Figure 6 shows how questions were created for the Pediatrics department and, utilizing Grok's multi-turn capability, obtaining the same JSON schema for the Emergency Department questions as well. This process was repeated until all six departments had sample data for their knowledge bases.



Provide 5 more inquiries and their responses but instead of questions to ask a cardiologist's office make them inquires you would ask a pediatrician's office.

```json
[
  {
    "inquiry": "What is the schedule for childhood vaccinations?",
    "response": "Vaccinations follow the CDC schedule, including DTaP at 2, 4, and 6 mon
  },
  {
    "inquiry": "How do I know if my child has a fever that needs medical attention?",
    "response": "For infants under 3 months, any fever over 100.4°F requires immediate a
  },
  {
    "inquiry": "Can you see my child for a same-day sick visit?",
    "response": "Yes, we offer same-day appointments for urgent concerns. Call us early
  },
  {
    "inquiry": "What should I do if my child is having trouble sleeping?",
    "response": "Establish a consistent bedtime routine, limit screen time before bed, a
  },
  {
    "inquiry": "Do you offer developmental screenings for toddlers?",
    "response": "Yes, we provide developmental screenings at well-child visits, typicall
  }
]
```

And now five inquires for the emergency department.

```json
[
  {
    "inquiry": "What should I do if I suspect someone is having a stroke?",
    "response": "Call 911 immediately. Note the time symptoms started and look for signs
  },
  {
    "inquiry": "How long is the current wait time in the emergency department?",
    "response": "Wait times vary based on patient volume and severity. Call our triage l
  },
  {
    "inquiry": "Can I bring a family member with me to the emergency room?",
    "response": "Yes, one family member is typically allowed, depending on the situation
  },
  {
    "inquiry": "What documents or information should I bring to the emergency department
    "response": "Bring your ID, insurance card, list of medications, and any relevant me
  },
  {
    "inquiry": "Do you treat non-life-threatening injuries like a sprained ankle?",
    "response": "Yes, we treat non-emergent injuries like sprains. However, urgent care
  }
]
```

Figure 6: Grok was used to generate knowledge base entries in JSON format.

The knowledge base for a given department is loaded into context when the inquiry first arrives at that agent. This can be seen line 18 of cell 6 (see figure 4.) If the inquiry is coming from the operator the department-specific `SystemMessage` is injected into conversation history. This message is built dynamically using the `get_system_message_for_agent` utility function defined in cell 4. It uses other abstracted functions to load the proper JSON and inject it into the prompt. This is one of several ways to load documents into context. While this was simple to implement it has drawbacks. The department's knowledge base, in its entirety, is counting against our token limits (including the questions we are not using.) An improvement to this agent would be to introduce retrieval nodes that select the relevant questions from a document store in a separate step before generating the response. This would take weight off of our queries and conversation history.

Requirement 7: Trial runs

There are six prompts required for this assignment. All six run successfully and return expected responses (an answer based on a related knowledge base entry or a message stating no such knowledge base entry exists.) However, this does not showcase all of the features in these agents (notably, multi-turn conversations.) Because of the interactive nature of this product it is difficult to show all permutations of how to use the bot in a written report. The recorded video for this assignment contains many trials and various permutations of agent behavior. It is stored on my NU Google Drive and can be accessed here:
`https://drive.google.com/file/d/1IsNPPwMN7NijN8GKMiPsc1RXaHa7gqhH/view`

Cell 15 shows how the application loop is constructed. It is then utilized for one of the trial prompts. In cell 15's output we demonstrate a multi-turn conversation. Some of the inquires match knowledge base context and some do not. We can also see the completion check working when the agent acknowledges the user is done. I will conclude with the transcript (figure 7) of one more trial, invoking several agent features.

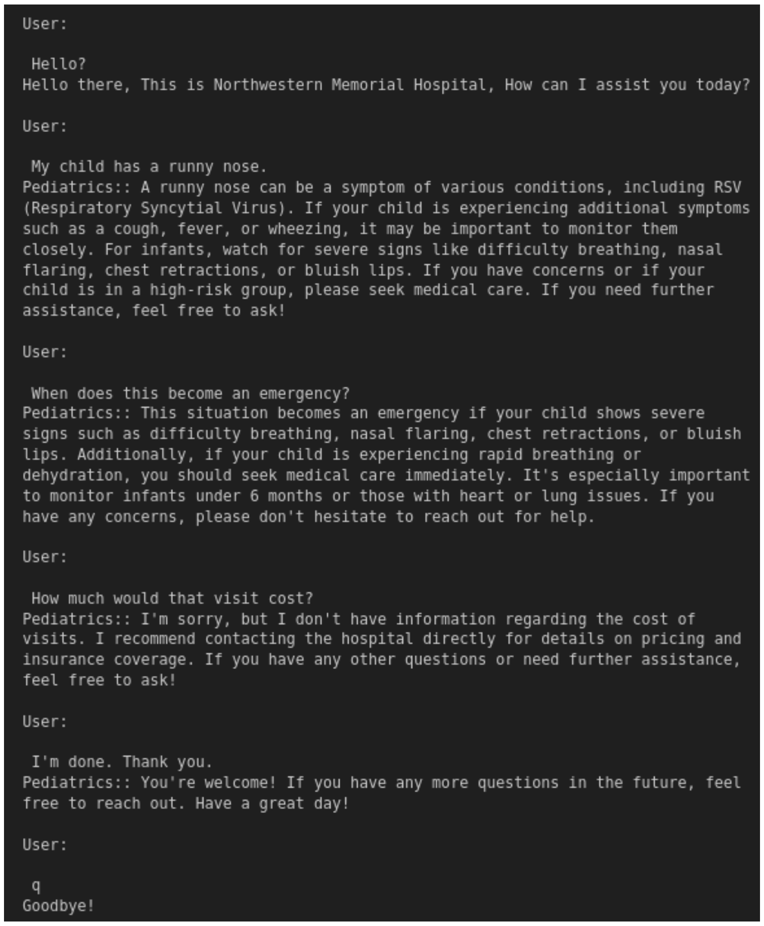

```
User:

 Hello?
Hello there, This is Northwestern Memorial Hospital, How can I assist you today?

User:

 My child has a runny nose.
Pediatrics:: A runny nose can be a symptom of various conditions, including RSV
(Respiratory Syncytial Virus). If your child is experiencing additional symptoms
such as a cough, fever, or wheezing, it may be important to monitor them
closely. For infants, watch for severe signs like difficulty breathing, nasal
flaring, chest retractions, or bluish lips. If you have concerns or if your
child is in a high-risk group, please seek medical care. If you need further
assistance, feel free to ask!

User:

 When does this become an emergency?
Pediatrics:: This situation becomes an emergency if your child shows severe
signs such as difficulty breathing, nasal flaring, chest retractions, or bluish
lips. Additionally, if your child is experiencing rapid breathing or
dehydration, you should seek medical care immediately. It's especially important
to monitor infants under 6 months or those with heart or lung issues. If you
have any concerns, please don't hesitate to reach out for help.

User:

 How much would that visit cost?
Pediatrics:: I'm sorry, but I don't have information regarding the cost of
visits. I recommend contacting the hospital directly for details on pricing and
insurance coverage. If you have any other questions or need further assistance,
feel free to ask!

User:

 I'm done. Thank you.
Pediatrics:: You're welcome! If you have any more questions in the future, feel
free to reach out. Have a great day!

User:

 q
Goodbye!
```

Figure 7: An example session that invokes multiple agent features.

[1]:
```
##########################################################
# MSDS 442: AI Agent Design and Development
# Spring '25
# Dr. Bader
#
# Assignment 4 - Northwestern Memorial - Healthcare Agent
#
# Kevin Geidel
#
##########################################################

# OBJECTIVE:
#   The following will construct multiple AI agents using the LangChain &␣
↪LangGraph frameworks.
#   The agents will represent different departments of Northwestern Memorial␣
↪Hospital.
#   They will coordinate, synchronize, and act to answer patients'/visitors'␣
↪questions.

# Load environment variables
from dotenv import load_dotenv
load_dotenv()

# Python native imports
import os, textwrap, json

# 3rd party package imports
from IPython.display import display, Image
from typing import Annotated, Sequence
from typing_extensions import TypedDict
from langgraph.graph import StateGraph, START, END
from langgraph.checkpoint.memory import MemorySaver
from langchain_core.messages import BaseMessage, HumanMessage, SystemMessage
from langchain_openai import ChatOpenAI

# Assign experiment-wide variables
model_name = 'gpt-4o-mini'
data_dir = os.path.join('reports', 'Assignment_4')
knowledge_base_dir = os.path.join('knowledge_base')
```

[2]:
```
# Requirement 1: Define the structure of agent state for the LangGraph
class InquiryState(TypedDict):
    inquiry: str
    referring_node: str
    next_node: str
    response: str
    messages: Annotated[Sequence[BaseMessage], "List of messages in the␣
↪conversation"]
```

[3]:
```
# Establish the AI client
llm = ChatOpenAI(model=model_name, temperature=0)
```

[4]:
```
# Define utils needed by the agents

def load_knowledge_base(filename):
    # Extract inquires and responses from the JSON format knowledge base
    full_path = os.path.join(knowledge_base_dir, filename)
    with open(full_path, 'r') as file:
        data = json.load(file)
    return str(data)

def get_query_from_inquiry(inquiry, messages=None):
    prompt_str = f"""Provide an answer for following user's inquiry: '{inquiry}'␣
↪using the knowledge_base."""
    if messages:
        history = "\n".join([f'{msg.type}: {msg.content}' for msg in messages][:
↪5])
        prompt_str += f"""\n\nConversation history for context:\n\n{history}"""
    return prompt_str

def get_human_message_for_agent(state):
    # Return the "HumanMessage" that forwards the user's inquiry (or last␣
↪agent's inquiry) to the next agent
    return HumanMessage(
            content=[
                {"type": "text", "text":␣
↪get_query_from_inquiry(state['inquiry'], state.get("messages", []))},
            ],
        )

def get_system_message_for_agent(knowledge_base_filename, department=None):
    return SystemMessage(
        content=f"You are a helpful assistant for the {department} department at␣
↪Northwestern Memorial Hospital. Answer the user's inquiry based solely on the␣
↪answers you have in this knowledge_base:␣
↪{load_knowledge_base(knowledge_base_filename)}\n\nUse the conversation history␣
↪to provide contextually relevant responses. If the user says 'quit' or␣
↪indicates the conversation is complete, respond appropriately and signal the␣
↪end. Otherwise, continue the conversation within the {department} department."
    )
```

```
[5]: def operator_router(state):
    inquiry = state['inquiry'].lower()
    messages = state.get("messages", [])

    # Check for end of conversation
    if inquiry in ['q', 'quit']:
        return {
            "inquiry": state["inquiry"],
            "referring_node": "Operator",
            "next_node": END,
            "response": "Goodbye! Thank you for contacting Northwestern Memorial!
↪",
            "messages": messages + [HumanMessage(content=inquiry),␣
↪SystemMessage(content="Conversation ended by user.")]
        }

    # Check for an ongoing conversation
    if state.get('referring_node') != "Operator" and state.get('next_node'):
        history = "\n".join([f"{msg.type}: {msg.content}" for msg in messages][:
↪5])
        query = f"""Given the conversation history and the new inquiry:␣
↪'{inquiry}', determine if this is a follow-up question related to the previous␣
↪department ({state['referring_node']}) or a new topic. Return 'continue' if␣
↪it's a follow-up, or classify the intent for a new topic.
        Possible intent values: Greeting, GeneralInquiry, ER, Radiology,␣
↪PrimaryCare, Cardiology, Pediatrics, BillingInsurance

        Conversation history:
        {history}
        """
        messages_for_intent = [
            SystemMessage(content="You are a helpful assistant tasked with␣
↪classifying the intent of a user's query or detecting follow-ups."),
            HumanMessage(content=[{'type': 'text', 'text': query}])
        ]
        response = llm.invoke(messages_for_intent)
        intent = response.content.strip()
        if intent == 'continue':
            return {
                "inquiry": state["inquiry"],
                "referring_node": "Operator",
                "next_node": state['referring_node'],
                "response": f"Continuing with the {state['referring_node']}␣
↪department.",
                "messages": messages + [HumanMessage(content=inquiry)]
            }

    # This is a new conversation. Have the operator decide how to route.
    query = f"""Classify the user's intents based on the following input:␣
↪'{state['inquiry']}'.
            List of possible intent values: Greeting, GeneralInquiry, ER,␣
↪Radiology, PrimaryCare, Cardiology, Pediatrics, BillingInsurance
            Return only the intent value of the inquiry identified with no extra␣
↪text or characters"""
    messages = [
        SystemMessage(content="You are a helpful assistant tasked with␣
↪classifying the intent of user's inquiry"),
        HumanMessage(content=[{"type": "text", "text": query}]),
    ]
    response = llm.invoke(messages)
    intent = response.content.strip()

    response_lower = intent.lower()

    if "greeting" in response_lower:
        response = "Hello there, This is Northwestern Memorial Hospital, How can␣
↪I assist you today?"
        next_node = END
    elif "generalinquiry" in response_lower:
        response = "For general informtion about nearby parking, hotels and␣
↪restaurants, please visit https://www.nm.org/ and navigate to Patients &␣
↪Visitors link "
        next_node = END
    else:
        response = f"Let me forward your query to our {intent} agent."
        next_node = intent

    return {
        "inquiry": state["inquiry"],
        "referring_node": "Operator",
        "next_node": next_node,
        "response": response,
    }
```

```
[6]: def department_specific_agent(state, department_node_name,␣
     ↪knowledge_base_filename):
         # Handle inquires related to the passed department
         inquiry = state['inquiry'].lower()
         messages = state.get("messages", [])

         # Check for end of conversation
         if inquiry in ['q', 'quit']:
             return {
                 "inquiry": state["inquiry"],
                 "referring_node": department_node_name,
                 "next_node": END,
                 "response": f"Goodbye! Thank you for contacting␣
     ↪{department_node_name} at Northwestern Memorial!",
                 "messages": messages + [HumanMessage(content=inquiry),␣
     ↪SystemMessage(content="Conversation ended by user.")]
             }

         if state['referring_node'] == 'Operator':
             # This is first pass at the department agent, include the system message
             messages += [get_system_message_for_agent(knowledge_base_filename)]+
             [get_human_message_for_agent(state)]
         else:
             # This is an ongoing conversation. Just append new inquiry
             messages += [get_human_message_for_agent(state)]

         response = llm.invoke(messages)
         formatted_response = f"{department_node_name}:: " + response.content.strip()

         # Check if conversation is over (next_node=END) or not (next_node=same␣
     ↪department)
         completion_check = llm.invoke([
             SystemMessage(content=f"Determine if the user's inquiry is fully␣
     ↪resolved based on the response: '{response.content}'.\n\nReturn 'complete' if␣
     ↪resolved, 'continue' if further interaction is needed."),
             HumanMessage(content=[{'type': 'text', 'text': f'Response: {response.
     ↪content}'}])
         ])
         next_node = END if completion_check.content.strip() == 'continue' else␣
     ↪department_node_name

         return {
             "input": state["inquiry"],
             "referring_node": department_node_name,
             "next_node": next_node,
             "response": formatted_response,
             "messages": messages + [SystemMessage(content=formatted_response)]
         }
```

[7]:
```
def er_agent(state):
    # Handle inquires related to the ER department
    return department_specific_agent(state, 'ER', 'emergency.json')
```

[8]:
```
def radiology_agent(state):
    # Handle inquires related to the Radiology department
    return department_specific_agent(state, 'Radiology', 'radiology.json')
```

[9]:
```
def primary_care_agent(state):
    # Handle inquires related to the PrimaryCare department
    return department_specific_agent(state, 'PrimaryCare', 'primary_care.json')
```

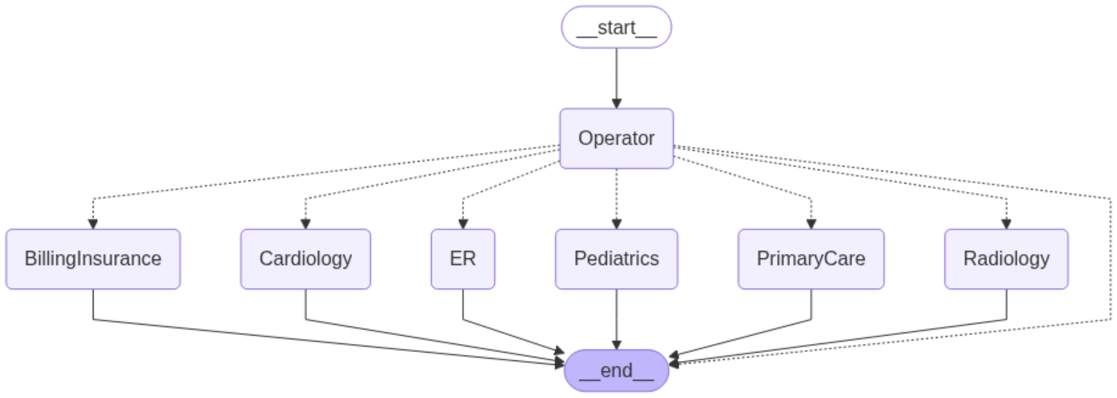
[10]:
```
def cardiology_agent(state):
    # Handle inquires related to the Cardiology department
    return department_specific_agent(state, 'Cardiology', 'cardiology.json')
```

[11]:
```
def pediatrics_agent(state):
    # Handle inquires related to the Pediatrics department
    return department_specific_agent(state, 'Pediatrics', 'pediatrics.json')
```

[12]:
```
def billing_agent(state):
    # Handle inquires related to the BillingInsurance department
    return department_specific_agent(state, 'BillingInsurance', 'billing.json')
```

[13]:
```
builder = StateGraph(InquiryState)

builder.add_node("Operator", operator_router)
builder.add_node("ER", er_agent)
builder.add_node("Radiology", radiology_agent)
builder.add_node("PrimaryCare", primary_care_agent)
builder.add_node("Cardiology", cardiology_agent)
builder.add_node("Pediatrics", pediatrics_agent)
builder.add_node("BillingInsurance", billing_agent)

builder.set_entry_point("Operator")

builder.add_conditional_edges(
    "Operator",
    lambda x: x["next_node"],
    {
        "ER": "ER",
        "PrimaryCare": "PrimaryCare",
        "Pediatrics": "Pediatrics",
        "Radiology": "Radiology",
        "Cardiology": "Cardiology",
        "BillingInsurance": "BillingInsurance",
        END: END
    }
)

for node in ["ER", "Radiology", "PrimaryCare", "Cardiology", "Pediatrics",␣
 ↪"BillingInsurance"]:
    builder.add_edge(node, END)


graph = builder.compile(checkpointer=MemorySaver())
```

[14]:
```
display(Image(graph.get_graph().draw_mermaid_png()))
```



[15]:
```
config = {"configurable": {"thread_id": "1"}}
while True:
    user_input = input("User: ")
    print(f"\nUser:\n\n {user_input}")
    if user_input.lower() in {"q", "quit"}:
        print("Goodbye!")
        break
    result = graph.invoke({"inquiry": user_input}, config=config)
    response = result.get("response", "No Response Returned")
    print(response)
```

```
User:

 How should I prepare for a CT scan and are there any dietary restrictions?
Radiology:: For a CT scan, you should fast for 4-6 hours if contrast dye is
used; clear liquids may be allowed. It's important to remove metal objects and
wear loose clothing. During the scan, you will need to stay still. Additionally,
inform your doctor about any allergies or kidney issues. Be sure to follow any
specific instructions provided, as some scans may have no dietary restrictions.

User:

 Can I schdule an appointment for one?
Radiology:: To schedule an appointment for a CT scan, you will need a referral
from your physician. Please contact our scheduling department to confirm and
book your appointment.

User:

 How much will a CT scan cost?
Radiology:: I'm sorry, but I don't have information regarding the cost of a CT
scan. I recommend contacting our billing department or your insurance provider
for specific cost details. If you have any other questions, feel free to ask!

User:

 That is all for me, thank you.
Radiology:: You're welcome! If you have any more questions in the future or need
assistance, feel free to reach out. Have a great day!

User:

 q
Goodbye!
```