Жамбыл облысы әкімдігі білім басқармасы

«Білім» кәсіби гуманитарлық-техникалық колледжі

(білім беру ұйымының атауы)

**Оқу сабағының жоспары**

(теориялық немесе өндірістік оқыту)

**Алгоритмнің трассировкасын жүзеге асыру.**

(сабақ тақырыбы)

**Модуль/Пән атауы** Бағдарламалық кодты қайта өңдеу

**Дайындаған педагог** Нургисаева У.М

**20\_**25**\_ жылғы** «\_\_\_\_» \_\_\_ақпан\_\_\_

**1. Жалпы мәліметтер**

Курс, оқу жылы, топ 3 курс, 3БҚ-22

Сабақ түрі: Тәжірибелік

**2. Мақсаты, міндеттері:**

**Оқу:** Білім алушыларға python (пайтон) программалау тілінде алгоритмнің трассировкасын жүзеге асыру, қолдану жолдарын ұйымдастыру, программаны өңдеу жұмыснегіздерін үйрету, шартты операторлардың жұмыс істеу принциптерін түсіндіру.

#### Дамыту: Логикалық ойлау қабілеттерін дамыту, есептердің әртүрлі шешімдерін табуға дағдыландыру, программалық кодтың құрылымын дұрыс құру қабілеттерін қалыптастыру.

#### Тәрбиелік: Жауапкершілік пен ұқыптылыққа тәрбиелеу, алгоритмдік ойлауды дамыту, шығармашылық қабілеттерін жетілдіру.

**3. Оқу-жаттығу процесінде білім алушылар меңгеретін күтілетін нәтижелер және кәсіби дағдылар тізбесі:** Python программалау тілінде алгоритмнің трассировкасын жүзеге асыру, қолдану жолдарын ұйымдастыру, программаны өңдеу және пайдалану дағдыларын меңгеру. Else циклін басқару, қолдану жолдарының жұмыснегіздерін үйренеді, код жазады.

**4. Қажетті ресурстар:** ДК немесе ноутбуктер

***5. Сабақтың барысы: (90 минут)***

**5.1. Ұйымдастыру кезеңі:** *( 3 мин )*

**5.2. Үй жұмысын жан-жақты тексеру:**

***“Домино ” әдісі*** *(15 минут)*

**Алгоритмнің трассировкасын жүзеге асыру.**

**Трассировка** - бағдарламаны қадамдар арқылы орындау процесі. Трассировка уақытында бағдарламашы командалардың орындалу ретін тексере алады және осы қадамда айнымалының қалай өзгеретінін көре алады. Бұл әдіс қателерді оңай табуға мүмкіндік береді.

**.3 Трассировканы жүзеге асыру әдістері**

Трассировка әртүрлі тәсілдер арқылы жүзеге асырылады:

**1) Қолмен трассировка жасау**

Қолмен трассировка – алгоритмді қағаз бетінде немесе электронды кестеде орындап, әр қадамдағы өзгерістерді тіркеу әдісі. Бұл әдіс шағын алгоритмдер үшін тиімді.

Мысал:  
Келесі **x** айнымалысын екі еселеу алгоритмінің трассировкасын жүргізейік.

x = 1

for i in range(4):

x = x \* 2

print(f"Қадам {i+1}: x = {x}")

Қолмен трассировка нәтижесі:

| **Қадам** | **x мәні** |
| --- | --- |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

**2) Логтау (журналға жазу) әдісі**

Көп жағдайда, бағдарламаның трассировкасын **логтар** арқылы бақылау тиімді. Логтау үшін келесі тәсілдер қолданылады:

* **Кәдімгі шығару операторы (print, println, cout т.б.)**
* **Логгер кітапханалары (Python-дағы logging, Java-дағы Logger және т.б.)**

Мысалы, Python бағдарламасында логтау қолдану:

import logging

logging.basicConfig(level=logging.DEBUG)

x = 1

for i in range(4):

x = x \* 2

logging.debug(f"Қадам {i+1}: x = {x}")

Бұл код орындалғанда DEBUG деңгейіндегі хабарламалар шығады.

**3) Дебаггер (бағдарламаны жөндеуші құралдар)**

Көптеген бағдарламалау орталарында (IDLE, PyCharm, Visual Studio Code, IntelliJ IDEA) **дебаггер** құралдары бар. Дебаггер көмегімен сіз:

* Бағдарламаны қадам бойынша орындап, айнымалылардың мәндерін көре аласыз.
* Белгілі бір жерге "үзіліс нүктесін" (breakpoint) қойып, бағдарламаны сол жерде тоқтатып, мәндерді талдай аласыз.

**4.4 Трассировканы қалай тиімді жүргізуге болады?**

Трассировка жасағанда келесі принциптерді ұстанған дұрыс:

* **Айнымалыларды анық бақылау.** Маңызды айнымалыларды ғана трассировкаға қосу.
* **Қадамдар арасындағы өзгерістерді көрсету.** Әрбір қадамда қандай өзгерістер болатынын түсіндіру.
* **Қателерді тауып, оңтайландыру жолдарын іздеу.** Қай жерде артық есептеулер барын анықтау.

**Дебаггер (Debugger)** – бұл бағдарламалық жасақтаманы талдау және жөндеу үшін қолданылатын арнайы құрал. Ол бағдарламаның орындалу процесін қадағалауға, қателерді анықтауға және түзетуге көмектеседі.

Бағдарламалар күрделене түскен сайын, олардағы қателерді (багтарды) табу қиындай түседі. Дебаггерді қолдану арқылы сіз кодтың әр қадамын тексеріп, айнымалылардың мәндерін көре аласыз, сондай-ақ бағдарламаны қажетті орындарда уақытша тоқтатып (breakpoint), мәндерді талдай аласыз.

**2. Дебаггердің негізгі мүмкіндіктері**

Дебаггер келесі негізгі функцияларды орындайды:

1. **Бағдарламаны қадам бойынша орындау (Step-by-step execution)**
   * Кодтың әр жолын біртіндеп орындауға мүмкіндік береді.
   * Әр қадамда айнымалылардың мәндерін тексеруге болады.
2. **Үзіліс нүктелерін (Breakpoint) орнату**
   * Бағдарлама белгілі бір жолға жеткен кезде автоматты түрде тоқтатылады.
   * Бұл кодтың қажетті бөлігін егжей-тегжейлі зерттеуге көмектеседі.
3. **Айнымалылардың мәндерін бақылау (Variable Inspection)**
   * Код орындалу барысында айнымалылардың қалай өзгеретінін көруге болады.
   * Бұл логикалық қателерді (мысалы, күтілген мәннің өзгермеуі) анықтауға көмектеседі.
4. **Шартты үзіліс нүктелері (Conditional Breakpoints)**
   * Белгілі бір шарт орындалғанда ғана бағдарламаны тоқтатады.
   * Мысалы, егер x == 5 болғанда ғана тоқтау керек болса, осындай шарт қоюға болады.
5. **Кері трассировка (Call Stack Analysis)**
   * Бағдарламаның қай функциядан шақырылғанын көруге мүмкіндік береді.
   * Бұл рекурсивті алгоритмдерде немесе күрделі логикаларда өте пайдалы.

**3.Дебаггерді қолдану мысалы**

**Python-дағы дебаггер (pdb модулі)**  
Python-да **pdb (Python Debugger)** атты кірістірілген дебаггер бар.

Мысалы, келесі бағдарламаны қарастырайық:

def multiply(a, b):

result = a \* b

return result

x = 5

y = 3

z = multiply(x, y)

print(f"Нәтиже: {z}")

Егер осы кодтың ішінде ақау бар деп ойласақ, pdb көмегімен оны тексеруге болады:

import pdb

def multiply(a, b):

pdb.set\_trace() # Осы жерде бағдарламаны тоқтату

result = a \* b

return result

x = 5

y = 3

z = multiply(x, y)

print(f"Нәтиже: {z}")

Бұл код орындалғанда pdb.set\_trace() жолында бағдарлама тоқтап, біз командалар арқылы әр қадамды зерттей аламыз.

**Басқару командалары:**

* n (next) – келесі жолға өту
* s (step) – функция ішіне кіру
* c (continue) – орындалуды жалғастыру
* p variable\_name – айнымалының мәнін қарау

**4.Графикалық интерфейсі бар дебаггерлер**

Көптеген бағдарламалау орталарында (IDE) кіріктірілген дебаггер бар:

1. **PyCharm (Python)**
   * Графикалық интерфейс арқылы дебаг жасауға болады.
   * Breakpoint орнатып, айнымалылардың мәнін көруге мүмкіндік береді.
2. **Visual Studio Code (Python, JavaScript, C++)**
   * Қарапайым және ыңғайлы дебаггері бар.
   * Chrome браузерімен интеграцияланған JavaScript дебаггері бар.
3. **IntelliJ IDEA (Java, Kotlin)**
   * Java және Kotlin үшін өте қуатты дебаггер ұсынады.
   * Thread (ағын) және Call Stack бақылауға болады.
4. **GDB (C, C++)**
   * Командалық жолда жұмыс істейтін қуатты дебаггер.
   * Бағдарламадағы қателерді төменгі деңгейде бақылауға мүмкіндік береді.

|  |  |  |
| --- | --- | --- |
| **Білдім** | **Білемін** | **Білгім келеді** |
|  |  |  |

**Рефлексия**

**Үй тапсырмасы –** Практикалық жұмыс №5 орындау және қорғау