Бюджетное учреждение высшего образования

Ханты-Мансийского автономного округа – Югры

**«Сургутский государственный университет»**

Отчет по лабораторной работе № 3

**«ПРЯМАЯ ЦЕПОЧКА РАССУЖДЕНИЙ»**

**Выполнил:**

Студент 3-ого курса

Гр. 607-91

Тунян Э.Г.

**Проверил:**

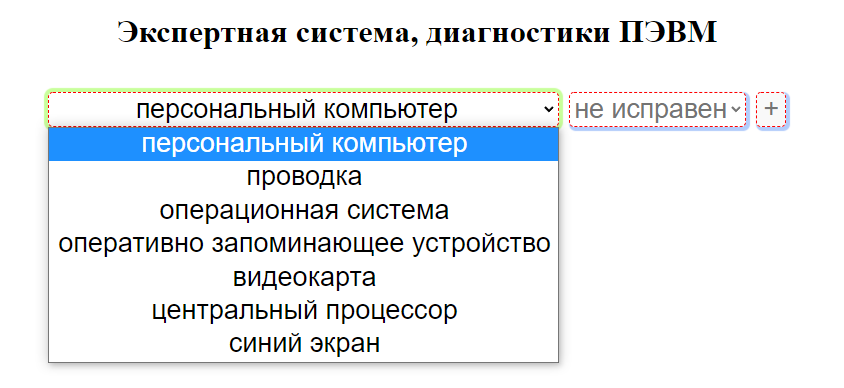
Шайторова И. А.

**2022 г.**

**Цель работы:**

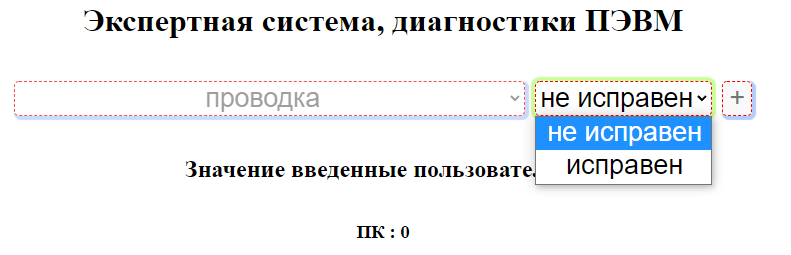
Разработать экспертную систему.

В начале пользователь отвечает на вопросы ЭС указывая с какими частями ПЭВМ были проблемы. Он может выбрать из предложенного списка переменную и задает значение либо «исправен», либо «не исправен».



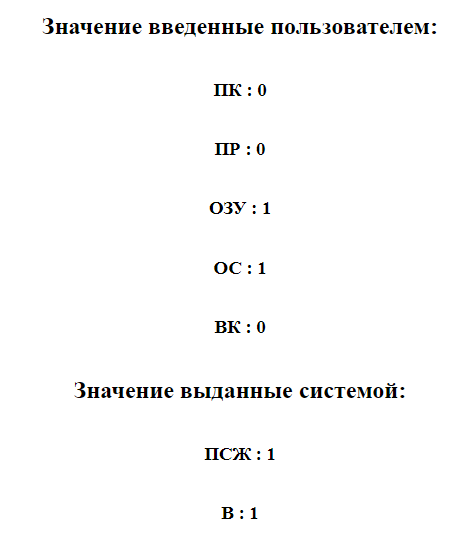
*Рис. 1. Этап выбора переменной*

Далее экспертная система запрашивает у пользователя недостающие переменные.



*Рис. 2. Ввод пользователем недостающих переменных*

После того, как пользователь ответил на все вопросы, экспертная система делает вывод.



*Рис. 3. Вывод экспертной системы*

1. Основная логика и код:

*import* { useEffect, useState, useRef } *from* "react";

*import* { category, regulation, abbreviations } *from* "./components/data";

*import* './styles/css/app.css'

*let* tRULE = {}

*function* App() {

*const* [flag, setFlag] = useState(true)

*const* [newAnsewerUser, setNewAnsewerUser] = useState([])

*const* [newAnsewerApp, setNewAnsewerApp] = useState([])

*const* [sel, setSel] = useState([])

*const* [rule, setRule] = useState({})

*const* inputAb = useRef()

*const* inputValue = useRef()

*const* \_regulation = regulation

*const* \_category = Array.from(category)

  useEffect(() *=>* {

*const* arr = []

    \_regulation.forEach(array *=>* {

      Object.keys(array).forEach(element *=>* {

        arr.push(<h3 *key*={Math.*random*()}>{element} : {array[element]}</h3>)

      })

      arr.push(<h3 *key*={Math.*random*()}>--------------</h3>)

    })

    category.forEach(el *=>* {

*const* [\_k, \_v] = Object.entries(el)[0]

*if* (\_k === "user") {

*const* \_ab = abbreviations.find(el *=>* Object.keys(el)[0] === \_v)

        setSel(oldEl *=>* [...oldEl, <option *key*={Math.*random*()} *value*={\_v}>{\_ab[\_v]}</option>])

      }

    })

  }, [\_regulation])

*const* ruleCheck = (\_allRules, rul) *=>* {

*/\*\**

*\* function takes all user rules and*

*\* and checks the current rule against*

*\*/*

*for* (*const* [\_key, \_value] of Object.entries(\_allRules)) {

*for* (*const* [rKey, rValue] of Object.entries(rul)) {

*if* (rKey === \_key)

*if* (*parseInt*(rValue) !== *parseInt*(\_value))

*return* false

      }

    }

*return* true

  }

*const* getType = (ab) *=>* {

*/\*\**

*\* function takes anabbreviation as a parameter and*

*\* returns type of abbreviation*

*\*/*

*const* index = category.findIndex(el *=>* Object.values(el).toString() === ab.toString())

*return* index > -1 *?* Object.keys(category[index]).toString() *:* false

  }

*const* checkFinish = (\_appKey, \_myRul) *=>* {

*/\*\**

*\* function takes a key and user rule and*

*\* checks whether the application is terminated or not*

*\*/*

*const* \_delArr = []

*if* (\_regulation.length > 0) {

*for* (*let* i = 0; i < \_category.length; i++) {

*const* [\_key, \_value] = Object.entries(\_category[i])[0]

*if* (\_key === "user" || \_appKey === \_value) \_delArr.push(i)

      }

    }

    \_delArr.sort((a, b) *=>* b - a)

    \_delArr.forEach(i *=>* \_category.splice(i, 1))

*const* key = Object.values(\_category[0])

*for* (*let* i = 0; i < \_regulation.length; i++) {

*const* arrKeyRegulations = Object.keys(\_regulation[i])

*if* (arrKeyRegulations.findIndex(e *=>* e == key) !== -1) {

        tRULE = \_regulation[i]

        \_regulation.splice(i, 1)

*break*;

      }

    }

    checkFillingRule(\_myRul)

  }

*const* getAppKey = (\_appKey, \_myRul) *=>* {

*/\*\**

*\* function takes a key and user rule and*

*\* push new application key and value*

*\*/*

*if* (ruleCheck(\_myRul, tRULE)) {

*const* \_obj = Object.entries(tRULE)

*const* oKey = \_obj[\_obj.length - 1][0]

*const* oValue = \_obj[\_obj.length - 1][1]

*const* oBject = { [oKey]: oValue }

      setRule(old\_o *=>* ({ ...old\_o, ...oBject }))

      setNewAnsewerApp(item *=>* [...item, <h3 *key*={Math.*random*()}>{oKey} : {oValue}</h3>])

      checkFinish(\_appKey, { ...\_myRul, ...oBject })

    } *else* {

*for* (*let* i = 0; i < \_regulation.length; i++) {

*const* arrKeyRegulations = Object.keys(\_regulation[i])

*if* (arrKeyRegulations.findIndex(e *=>* e == \_appKey) !== -1) {

*if* (ruleCheck(\_myRul, \_regulation[i])) {

*const* \_obj = Object.entries(\_regulation[i])

*const* oKey = \_obj[\_obj.length - 1][0]

*const* oValue = \_obj[\_obj.length - 1][1]

*const* oBject = { [oKey]: oValue }

            setRule(old\_o *=>* ({ ...old\_o, ...oBject }))

            setNewAnsewerApp(item *=>* [...item, <h3 *key*={Math.*random*()}>{oKey} : {oValue}</h3>])

            checkFinish(\_appKey, \_myRul)

*break*

          }

        }

      }

    }

  }

*function* onInput(\_key) {

*/\*\**

*\* function takes a key and*

*\* enables the user to select the value of a variable*

*\*/*

    inputAb.current.value = \_key

    inputAb.current.disabled = true

*// inputValue.current.value = ""*

    inputValue.current.focus()

    setFlag(true)

  }

*const* seatchKey = (param) *=>* {

*/\*\**

*\* function takes a key as a parameter and*

*\* return index of the regulation that conains this key*

*\*/*

*let* \_flag = true

*let* temp

*let* \_delArr = []

*for* (*const* [key, value] of Object.entries(param)) {

*for* (*let* i = 0; i < \_regulation.length; i++) {

*const* arrKeyRegulations = Object.keys(\_regulation[i])

*if* (arrKeyRegulations.findIndex(e *=>* e == key) !== -1) {

*const* result = \_regulation[i]

*if* (result[key] === *parseInt*(value) && \_flag) {

            \_delArr.push(i)

            temp = result

            \_flag = false

          }

*else* *if* (*parseInt*(Object.values(\_regulation[i])) !== *parseInt*(value))

            \_delArr.push(i)

        }

      }

    }

    \_delArr.sort((a, b) *=>* b - a)

    \_delArr.forEach(i *=>* \_regulation.splice(i, 1))

*return* temp *?* temp *:* false

  }

*const* checkFillingRule = (\_myRul = rule) *=>* {

*/\*\**

*\* function check and filling rule*

*\*/*

*let* checkRes = ruleCheck(\_myRul, tRULE)

*do* {

      checkRes = ruleCheck(\_myRul, tRULE)

*if* (!checkRes) {

        tRULE = seatchKey(\_myRul)

      }

    } *while* (!checkRes);

*const* keys = Object.keys(tRULE)

*const* \_rL = keys.length

*for* (*let* i = 0; i < keys.length; i++) {

*const* \_key = keys[i]

*if* (Object.keys(\_myRul).findIndex(k *=>* k === \_key) === -1) {

*const* \_index = Object.keys(tRULE).findIndex((e) *=>* e === \_key)

*const* type = getType(\_key)

*if* (\_index === \_rL - 1 && type === "app") {

          getAppKey(\_key, \_myRul)

*break*;

        } *else* *if* (type === "user") {

          onInput(\_key)

*break*;

        }

      }

    }

  }

*const* searchRegulations = (arrKeys) *=>* {

*/\*\**

*\* function takes an array of objects as a parameter and*

*\* returns number (index) of the regulation that contains this key(s)*

*\*/*

    setRule(old\_o *=>* ({ ...old\_o, ...arrKeys }))

*if* (Object.keys(tRULE).length === 0) {

*const* res = seatchKey(arrKeys)

      tRULE = res

    }

    checkFillingRule({ ...rule, ...arrKeys })

  }

*const* calculate = () *=>* {

    setFlag(false)

*const* ab = inputAb.current.value.toUpperCase()

*const* val = inputValue.current.value

    console.clear()

    setNewAnsewerUser(item *=>* [...item, <h3 *key*={Math.*random*()}>{ab} : {val}</h3>])

    searchRegulations({ [ab]: val })

  }

*return* (

    <div *className*="app">

      {flag &&

        <div *className*="center">

          <h1>Экспертная система, диагностики ПЭВМ</h1>

          <h2 *className*="abTitle">

            <select *ref*={inputAb} *className*="select\_menu text\_center" >

              {sel}

            </select>

            {*/\* <input placeholder="Аб." className="ab" ref={inputAb} /> \*/*}

            {*/\* <input placeholder={"значение"} ref={inputValue} className="select\_menu" /> \*/*}

            <select *ref*={inputValue} *className*="select\_menu big">

              <option *value*={0}>не исправен</option>

              <option *value*={1}>исправен</option>

            </select>

            <button *onClick*={() *=>* calculate()} *className*="select\_menu">+</button>

          </h2>

        </div>

      }

      {

        newAnsewerUser.length > 0 && (

          <div *className*="center">

            <h2>Значение введенные пользователем:</h2>

            {newAnsewerUser}

          </div>

        )

      }

      {

        newAnsewerApp.length > 0 && (

          <div *className*="center">

            <h2>Значение выданные системой:</h2>

            {newAnsewerApp}

          </div>

        )

      }

    </div>

  );

}

*export* *default* App;

Вспомогательные массивы data.js:

*export* *const* regulation = [

    { "ОС": 0, "ПСЖ": 1 },

    { "ОС": 1, "ПСЖ": 0 },

    { "ПК": 0, "ПР": 0, "ПСЖ": 1 },

    { "ПСЖ": 0, "СОС": 1, "В": 1 },

    { "ПСЖ": 1, "ОЗУ": 1, "В": 1 },

    { "ПСЖ": 1, "ОЗУ": 1, "ВК": 1, "В": 0 },

    { "ПСЖ": 1, "ОЗУ": 1, "ВК": 1, "В": 1 },

    { "ПСЖ": 1, "ОЗУ": 1, "ВК": 1, "ЦП": 0, "В": 0 },

    { "ПСЖ": 1, "ОЗУ": 1, "ВК": 1, "ЦП": 1, "В": 1 },

    { "ПСЖ": 0, "ОС": 0, "СЭ": 0, "СОС": 0 },

    { "ПСЖ": 1, "ОС": 0, "СЭ": 1, "СОС": 1 },

    { "ПСЖ": 1, "ОЗУ": 1, "ВК": 1, "ЦП": 1, "В": 1 },

    { "ПК": 0, "ПР": 1, "ОС": 0, "ПСЖ": 1 },

]

*export* *const* abbreviations = [

    { "ПК": "персональный компьютер" },

    { "ПР": "проводка" },

    { "В": "вывод" },

    { "ОС": "операционная система" },

    { "ПСЖ": "проблема с железом" },

    { "ОЗУ": "оперативно запоминающее устройство " },

    { "ВК": "видеокарта" },

    { "ЦП": "центральный процессор" },

    { "СЭ": "синий экран" },

    { "СОС": "статус операционной системы" },

]

*export* *const* category = [

    { "app": "В" },

    { "app": "СОС" },

    { "app": "ПСЖ" },

    { "user": "ПК" },

    { "user": "ПР" },

    { "user": "ОС" },

    { "user": "ОЗУ" },

    { "user": "ВК" },

    { "user": "ЦП" },

    { "user": "СЭ" },

]