УДК 371.3

МЕЖДИСЦИПЛИНАРНЫЙ ПОДХОД ПРИ ИЗУЧЕНИИ ТИПОВ И СТРУКТУР ДАННЫХ В ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Симурзина Екатерина Анатольевна
аспирант
Научный руководитель: **Казиахмедов Туфик Багаутдинович**
к.п.н., доцент
ФГБОУ ВО «Нижневартовский государственный университет»

Аннотация: В представленной работе описан междисциплинарный подход при изучении типов и структур данных на межязыковой основе. В качестве среды разработки использованы C++, PascalABC, PyCharm, Java Script. В работе рассматривается изучение типов и структур как математических структур с особенностями их реализации в разных языках высокого уровня. Именно использование разных языков программирования дает полное представление об их реализации в языках программирования. Результатом работы является элементы методической системы междисциплинарного обучения типам и структурам данных, организации совместимости этих структур при разработке многоязычных программ.

Ключевые слова: классификация типов и структур данных; математические структуры; многоязычные проекты и приложения.

INTERDISCIPLINARY APPROACH TO STUDYING DATA TYPES AND STRUCTURES IN PROGRAMMING LANGUAGES.

Simurzina Ekaterina Anatolievna
Scientific adviser: **Kaziakhmedov Tufik Bagautdinovich**

Abstract: The presented work describes an interdisciplinary approach to the study of data types and structures on an interlanguage basis. C ++, PascalABC, PyCharm, Java Script were used as a development environment. The paper deals with the study of types and structures as mathematical structures with the peculiarities of their implementation in different high-level languages. It is the use of different programming languages that gives a complete picture of their implementation in programming languages. The result of the work is the elements

of a methodological system for interdisciplinary teaching of data types and structures, the organization of the compatibility of these structures in the development of multilingual programs.

Keywords: classification of data types and structures; mathematical structures; multilingual projects and applications.

Обучение IT бакалавров разработке ПО требует формирования у них понимания важности структурирования данных и их использования для решения задач автоматизации деятельности различных отделов предприятий. Порой сталкиваемся с тем, что данные собранные в одних программах, студенты с трудом используют в других, с трудом представляют их в разных форматах. Поэтому обучение типам и структурам данных нужно рассматривать как один из основных элементов обучения программированию. Причем управление потоками данных в разных языках программирования различно по своим возможностям [1]. Работа по эффективному использованию структур данных и потоков данных требует их всестороннего анализа с целью недопущения создания лишних копий.

Программируя решение любой задачи, необходимо выбрать уровень абстрагирования. Иными словами, определить множество данных, представляющих предметную область решаемой задачи. При выборе следует руководствоваться проблематикой решаемой задачи и способами представления информации. Здесь необходимо ориентироваться на те средства, которые предоставляют системы программирования и вычислительная техника, на которой будут выполняться программы. Во многих случаях эти критерии не являются полностью независимыми.

Изучая программирование, студенты должны иметь представление о возможностях разных языков в формировании математических структур данных. Говоря о математических структурах, необходимо отметить, что это составные данные: векторы, последовательности, множества, отображения, матрицы, тензоры, графы, деревья и т.д. (см. рис.1.).

Можно сказать, что структура данных – это кейс, хранящий данные в определенном макете, который позволяет структуре данных быть эффективной в некоторых операциях и неэффективной в других.

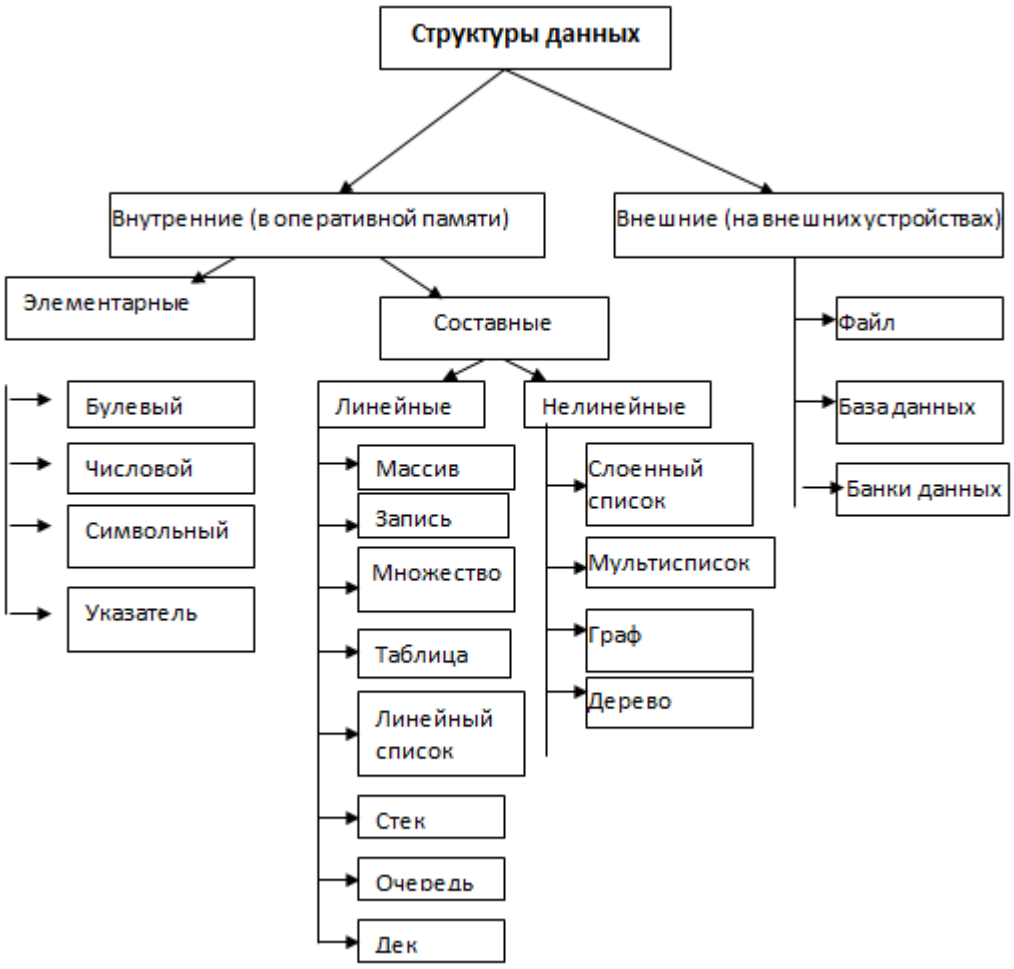

Рис. 1. Структуры данных

Полноценный подход в формировании всех этих структур можно реализовать на многоязычном программировании. О многоязычии при изучении программирования в первом курсе по направлению ИВТ сказано в [2].

Массивы – одна из самых широко используемых и самых простых структур данных. Задачи на массивах важны для изучения самих языков. В математике массивы – это вектора, матрицы, n-мерные массивы. В программировании важно, как и где хранить данные в этих структурах. Поэтому, говоря о последовательностях, следует обратить внимание на их связность или не связность. Однонаправленные, двунаправленные, кольцевые, мультисписки очень важны для формирования различных моделей памяти для хранения данных. Использование этих структур, которые в ЭВМ представляются в динамической (Heap) памяти позволяют решить задачи на графах, деревьях и т.д.

Наш опыт позволяет нам выполнить следующие сравнения языков высокого уровня (Таблица 1).

Таблица 1

Сравнение методов реализации составных структур

Тип данных и структур	Java Script	Python	C++
Вектор	Одномерный массив	Список[]	Одномерный массив, шаблон vector
Множества, Мультимножества	Представление в виде массива	Множество {}	Шаблоны Set,MultiSet
Отображения, отображения с дубликатами	Программная реализация	Список пар	Программная реализация, шаблоны map,multimap
Двумерный массив	Двумерный массив	Список списков	Двумерный массив
Списки однонаправленные, двунаправленные, циклические	На основе массивов	Списки	На основе указателей, шаблон List
Стек	Программная реализация	Класс stack	Программная реализация, шаблон List
Очередь	Программная реализация	Класс queue	Программная реализация через указатель *, шаблон queue
Дек	Программная реализация	Класс deque	Программная реализация через указатель *, шаблон deque
Таблица	Массив записей	Список кортежей	Массив записей, в том числе и динамический
Запись	Встроенный тип	Список	Встроенный тип(struct)
База данных	таблицы	Список картежей	таблицы
Граф	Программная реализация(матрица смежности,матрица инцидентности, список вершин, список ребер)	Вложенные списки	Программная реализация (матрица смежности,матрица инцидентности, список вершин, список ребер и др)
Дерево	Программная реализация (методы матрица смежности, матрица инцидентности, список вершин, список ребер, двунаправленный список)	Вложенные списки	Программная реализация (методы матрица смежности, матрица инцидентности, список вершин, список ребер, двунаправленный список)

Составные структуры нужно реализовать как абстракции, т.е. как данные + методы работы с данными.

Следовательно, при абстрагировании мы рассматриваем составные типы как математические структуры, объясняя связь реализуемых методов именно для каждой структуры с математикой: алгебра логики, теория множеств, матричная алгебра, теория графов, алгоритмы на графах и деревьях, алгоритмы поиска на графах и деревьях[3].

В программировании хранение данных в оперативной памяти выполняется как в статической, так и динамической памяти (Heap-область). Изучение особенностей использования динамической памяти лучше демонстрировать на примере безразмерных массивов разных типов. Ниже приводится код работы с динамическими массива на C++:

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
struct ms{
string fio;
int voz;
};
int* a,**bb;  float *b; ms *mas;
  int n;
void main(){
system("chcp 1251 >null");
cout<<"Введите  N"<<endl;
cin>>n;
//a=new int[n];
mas=new ms[n];
a=(int *)calloc(n,sizeof(int));
a[1]=7;
for (int i=0;i<n;i++)a[i]=i;
for (int i=0;i<n;i++)cout<<a[i]<<endl;
b=new float[n];
//b=(float *) calloc(n,sizeof(float));
for (int i=0;i<n;i++)*(b+i)=float(a[i]);
for (int i=0;i<n;i++)printf("\n%f",(*(b+i)));
mas[0].fio="Иванов";
```

```
mas[0].voz=30;
 mas[1].fio="Петров";
mas[1].voz=28;
   cout<<" "<<endl;
 for (int i=0;i<2;i++){cout<<mas[i].fio<<" "<<mas[i].voz<<endl;}
 getch();
}
```

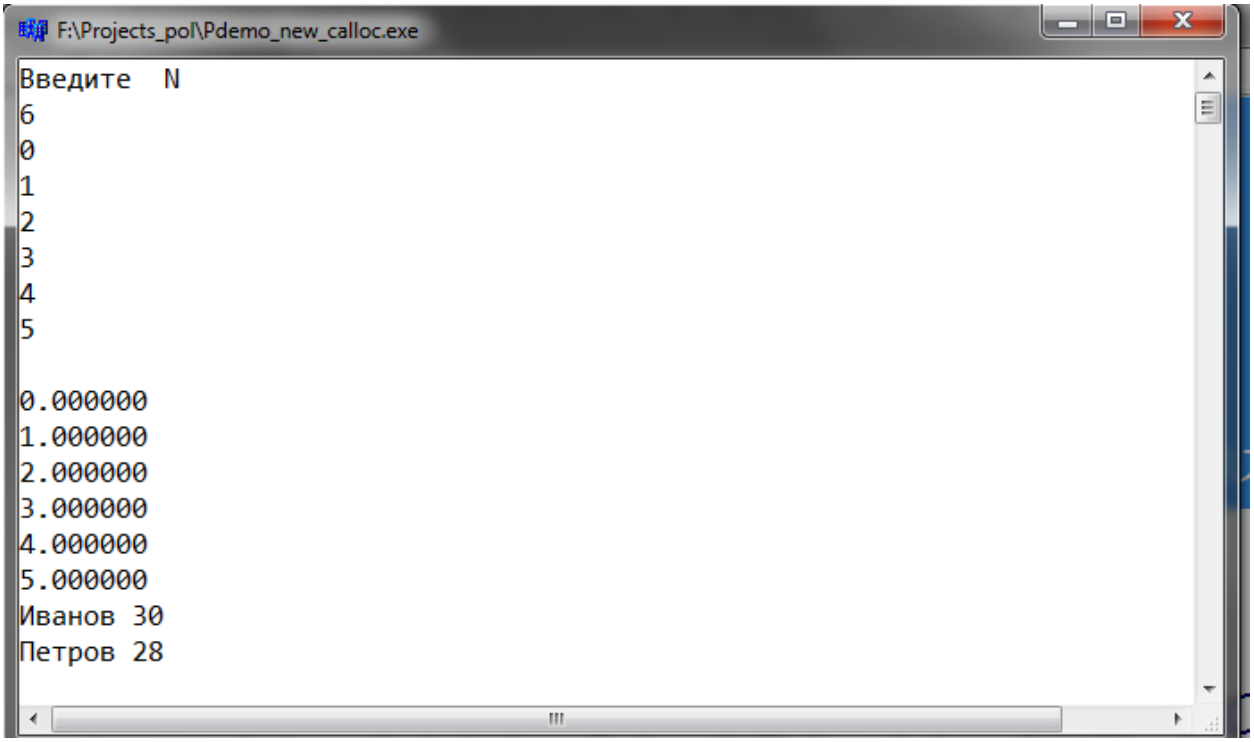

Рис.2 . Экран выполненной программы

Основной структурой для полного понимания того, как хранятся данные в памяти, является связанный список. В отличие от массивов связанные списки имеют структурную гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задается его внутренними связями. Реализация списков формирует и представление о программировании всех видов списков:

- однонаправленные: каждый узел хранит адрес или ссылку на следующий узел в списке и последний узел имеет следующий адрес или ссылку как NULL (1->2->3->4->NULL)
- двунаправленные: две ссылки, которые связаны с каждым узлом,

одним из опорных пунктов на следующий узел и один к предыдущему узлу (NULL<-1<->2<->3->NULL)

- круговые: все узлы соединяются и образуют круг. В конце нет NULL. Циклический связанный список может быть одно или двукратным циклическим связанным списком (1->2->3->1)

Для списков реализуется следующие основные операции:

- InsertAtEnd – вставка заданного элемента в конец списка
- InserAtHead – вставка элемента в начло списка
- Delete – удаляет заданный элемент из списка
- DeleteAtHead – удаляет первый элемент списка
- Search – возвращает заданный элемент из списка
- IsEmpty – возвращает true, если связанный список пуст

С использованием структур данных при разработке программ более подробно можно ознакомиться в [4].

Графы и деревья –это математические структуры, программирование которых ускоряет задачи поиска и сортировки, особенно задачи поиска в больших объемах данных. Граф как математический объект представляет собой совокупность двух множеств – множество самих объектов (вершин) и множество их парных связей (ребер). Графы находят широкое применение во многих современных сферах. Они используются и в социальных науках (например, социологии) и в естественных науках (физике и химии), но наибольшее применение графы получили в информатике и сетевых технологиях.

Пример программы реализации графов на C++:

```
//Обход графа в ширину.
#include <iostream>
#include <queue> // очередь
using namespace std;
int main()
{
  queue<int> Queue;
  int mas[7][7] = { { 0, 1, 1, 0, 0, 0, 1 }, // матрица смежности
  { 1, 0, 1, 1, 0, 0, 0 },
  { 1, 1, 0, 0, 0, 0, 0 },
  { 0, 1, 0, 0, 1, 0, 0 },
  { 0, 0, 0, 1, 0, 1, 0 },
```

```
{ 0, 0, 0, 0, 1, 0, 1 },
{ 1, 0, 0, 0, 0, 1, 0 } };
int nodes[7]; // вершины графа
for (int i = 0; i < 7; i++)
 nodes[i] = 0; // исходно все вершины равны 0
Queue.push(0); // помещаем в очередь первую вершину
while (!Queue.empty())
{ // пока очередь не пуста
 int node = Queue.front(); // извлекаем вершину
 Queue.pop();
 nodes[node] = 2; // отмечаем ее как посещенную
 for (int j = 0; j < 7; j++)
 { // проверяем для нее все смежные вершины
  if (mas[node][j] == 1 && nodes[j] == 0)
  { // если вершина смежная и не обнаружена
   Queue.push(j); // добавляем ее в очередь
   nodes[j] = 1; // отмечаем вершину как обнаруженную
  }
 }
 cout << node + 1 << endl; // выводим номер вершины
}
cin.get();
return 0;
}
```

Дерево – это иерархическая структура данных, которая состоит из узлов (вершин) и ребер (дуг). По сути, деревья – это связанные графы без циклов. Важно, чтобы студенты программировали деревья следующих видов:

- N- деревья
- Сбалансированные деревья
- Бинарные деревья
- Дерево бинарного поиска
- AVL дерево
- 2-3-4 деревья

Одно из самых распространенных – бинарное дерево.

Имеется три способа обхода деревьев:

1. В прямом порядке (сверху-вниз) - префиксная форма.

2. В симметричном порядке (слева-направо) – инфиксная форма
3. В обратном порядке (снизу-вверх) – постфиксная форма

Пример реализации дерева на C++:

```
#include <stdlib.h>
#include <iostream>
using namespace std;
struct node {
  int data;
struct node *left;
  struct node *right;
};
// Создание нового узла
struct node *newNode(int data) {
  struct node *node = (struct node *)malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;
  return (node);
}
// Прямой обход дерева
void traversePreOrder(struct node *temp) {
  if (temp != NULL) {
    cout << " " << temp->data;
    traversePreOrder(temp->left);
    traversePreOrder(temp->right);
  }
}
```

IT индустрия ставит серьезные проблемы перед вузами и государством в условиях информатизации всех сфер экономики государства [5].

Таким образом, подготовка IT специалистов становится одной из главных задач при подготовке кадров для цифровой экономики.

Список литературы

1. Казиахмедов Т.Б. Яламов Г.Ю. Содержание и методы обучения программированию бакалавров по направлению "информатика и вычислительная техника". Педагогическая информатика № 4, 2019. С. 47-58

2. Казиахмедов Т.Б., Симурзина Е. А. Междисциплинарный подход в обучении программированию бакалавров ИВТ // Материалы III Международной научно-практической конференции «Современное программирование». Нижневартовск, 2021. Издательство: Нижневартовский государственный университет. - С. 246-250

3. Казиахмедов Т.Б. Информатика как профессионально-ориентированная дисциплина в системе подготовки бакалавров // Материалы III Международной научно-практической конференции «Современное программирование». Нижневартовск, 2021.Издательство: Нижневартовский государственный университет. - С. 19-24

4. Казиахмедов Т.Б., Елыкомов Р.Н. Этапы разработки автоматизированных информационных систем удаленного тестирования обучающихся. Информатизация образования - 2020: Международная научно-практическая конференция, посвященная 115-летию со дня рождения патриарха российского образования, великого педагога и математика, академика РАН С. М. Никольского (1905 - 2012 гг.). МОО «Академия информатизации образования»; ОГУ имени И.С. Тургенева. Орел, 2020. - Издательство: Орловский государственный университет имени И.С. Тургенева. - С. 329-334

5. Казиахмедов Т.Б. Опережающее обучение в области индустрии информационных технологий в условиях развивающейся экономики и перманентных реформ высшего образования // Педагогическая информатика №4, 2014 год. С. 62-67