Тема 2. Статические структуры данных

Статические структуры относятся к разряду непримитивных структур, которые, фактически, представляют собой структурированное множество примитивных, базовых, структур.

Например, вектор может быть представлен упорядоченным множеством чисел.

Т.к. по определению статические структуры отличаются отсутствием изменчивости, память для них выделяется автоматически - как правило, на этапе компиляции или при выполнении - в момент активизации того программного блока, в котором они описаны.

Выделение памяти на этапе компиляции является столь удобным свойством статических структур, что в ряде задач программисты используют их даже для представления объектов, обладающих изменчивостью.

Например, когда размер массива неизвестен заранее, для него резервируется максимально возможный размер.

Каждую структуру данных характеризуют ее ***логическим*** и ***физическим представлениями***.
Говоря о той или иной структуре данных, имеют в виду ее логическое представление.
Физическое представление обычно не соответствует логическому, и может существенно различаться в разных программных системах.
Нередко физической структуре ставится в соответствие ***дескриптор***, или заголовок, который содержит общие сведения о физической структуре.

Дескриптор необходим, например, в том случае, когда граничные значения индексов элементов массива неизвестны на этапе компиляции, и, следовательно, выделение памяти для массива может быть выполнено только на этапе выполнения программы.

Дескриптор хранится как и сама физическая структура, в памяти и состоит из полей, характер, число и размеры которых зависят от той структуры, которую он описывает и от принятых способов ее обработки.

В ряде случаев дескриптор является совершенно необходимым, так как выполнение операции доступа к структуре требует обязательного знания каких-то ее параметров, и эти параметры хранятся в дескрипторе.
Другие хранимые в дескрипторе параметры не являются совершенно необходимыми, но их использование позволяет сократить время доступа или обеспечить контроль правильности доступа к структуре.
Дескриптор структуры данных, поддерживаемый языками программирования, является "невидимым" для программиста; он создается компилятором и компилятор же, формируя объектные коды для доступа к структуре, включает в эти коды команды, обращающиеся к дескриптору.

Статические структуры в языках программирования связаны со ***структурированными типами***. Структурированные типы в языках программирования являются теми средствами интеграции, которые позволяют строить структуры данных сколь угодно большой сложности.

К таким типам относятся:

- ♦ массивы,
- ♦ записи (в некоторых языках - структуры)
- ♦ и множества (этот тип реализован не во всех языках).

3.1. Массивы

Массивы являются наиболее широко используемыми структурами данных и предусмотрены во всех языках программирования.

Массив состоит из элементов одного типа, называемого ***базовым***, поэтому структура массива однородна.

Базовый тип может быть как ***скалярным***, так и ***структурированным***, т.е. элементами массива могут быть числа, символы, строки, структуры, в том числе и массивы.

Число элементов массива фиксировано, поэтому объем занимаемой массивом памяти, остается неизменным.

С каждым элементом массива связан один или несколько ***индексов***. Они однозначно определяют место элемента в массиве и обеспечивают прямой доступ к нему.
Индексы массива относятся к определенному ***порядковому*** типу, поэтому индексы можно вычислять.

Это обеспечивает, с одной стороны, гибкость обработки элементов массива, с другой стороны, создает опасность выхода за пределы массива, если не предусмотрены соответствующие средства контроля.

В зависимости от числа индексов различают ***одномерные*** и ***многомерные массивы***.
Допустимое число индексов массива (***размерность массива***) и диапазоны изменения их значений устанавливаются языком программирования, трансляторы с языка программирования могут уточнить эти значения.
По стандарту языка С размерность массива не может превышать 31, а нумерация индексов начинается всегда с нуля, т.е. индекс изменяется от 0 до n-1, где n - количество значений индекса (мощность индекса).

Областями применения массивов являются:

- числовые массивы в вычислительных задачах;
- матричная алгебра, экстраполяция, интерполяция;
- указатель (адрес) начала списка;
- таблицы - массивы с элементами типа «запись»;
- управляющие и информационные таблицы в операционных системах, трансляторах, системах управления базами данных (СУБД);
- представление других структур: графов, деревьев.

Вся информация, необходимая для управления массивом, задается при его ***описании в программе***.
Описание содержит ***имя массива***, ***тип элементов***, который однозначно определяет ***длину элемента***, ***диапазоны изменения индексов*** или ***число значений индексов***, если нижние границы индексов фиксированы.
Таким образом, общее количество элементов массива и размер памяти для массива полностью определяются описанием массива.

Синтаксис описания массива в Паскале представляется в виде:

< Имя > : Array [n1 .. k1] [n2 .. k2] ... [nn .. kn] of < Тип >;

Для случая двумерного массива:

Mas2D : Array [n1 .. k1] [n2 .. k2] of < Тип >;

или

Mas2D : Array [n1 .. k1, n2 .. k2] of < Тип >;

Транслятор выделяет необходимую память и строит управляющий блок массива - ***дескриптор***, или информационный вектор, например, такого содержания (для одномерного массива):

- тип структуры;
- адрес начала массива A_n;
- тип элемента (длина элемента массива L);
- нижняя граница индекса i_n;
- верхняя граница индекса i_k.

В случае многомерного массива для каждого индекса задаются нижняя и верхняя границы или же для каждой строки (каждого индекса) создается свой дескриптор.

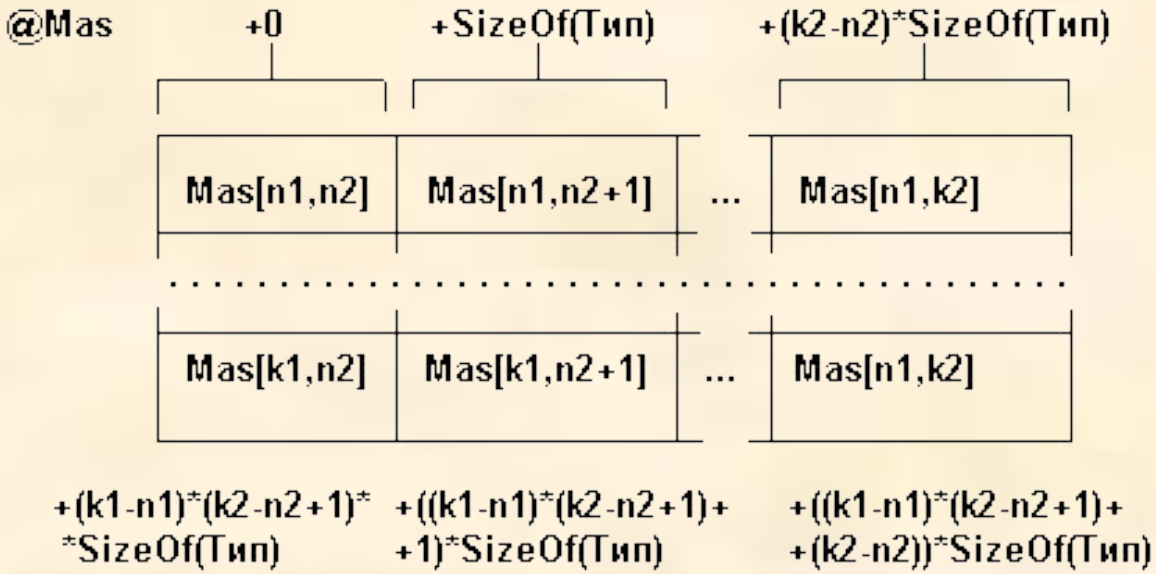
Этой информации достаточно как для доступа к элементам массива, так и для контроля над тем, чтобы значения индексов не выходили за установленные диапазоны.

Массив в памяти хранится в виде вектора, т.е. все элементы размещаются в смежных участках памяти подряд, начиная с адреса, соответствующего началу массива.

Элементы одномерного массива размещаются в последовательности A_0, A_1, A_2, …, A_{n-1}.

Элементы двухмерного массива размещаются либо по строкам, когда наиболее быстро меняется последний индекс (в С, Паскале, ПЛ/1), либо по столбцам, когда наиболее быстро меняется первый индекс (в Фортране).

Физическая структура двумерного массива из (k1-n1+1) строк и (k2-n2+1) столбцов:



Доступ к любому ***i***-му элементу в одномерном массиве осуществляется по адресу:

$$A_i = A_n + (i - i_n) * L = A_n - i_n * L + i * L = A_0 + i * L,$$

где $A_0 = A_n - i_n * L$ - фиктивное начало массива, т.е. адрес нулевого элемента.

То, что ***размеры массива***, формируемого транслятором, ***фиксированы***, может явиться ограничивающим фактором применения готовой программы.
Действительно, требуется, чтобы память для массива выделялась в размерах, необходимых для решения конкретной задачи, а каковы будут ее потребности, заранее может быть неизвестно.
В таких ситуациях массив можно строить в ***динамической памяти***, получаемой с помощью средств управления памятью операционной системы. Управление, доступом к элементам таких массивов осуществляется самой программой по ***вычисляемым индексам*** (адресам) ***элементов*** с использованием ***указателя***.

Пусть n-мерный массив A, у которого число элементов по I-М измерениям равно m_i, а индексация по всем измерениям начинается с нуля, представлен в динамической памяти в виде вектора B.

Тогда элементу $A[i_1, i_2, \ldots, i_n]$ исходного массива будет соответствовать элемент вектора B[к] с индексом, равным

$$k=((\ldots(i_1*m_2+i_2)*m_3+i_3)*m_4+\ldots+i_{n-1})*m_n+i_n.$$

Существуют различные варианты использования динамической памяти, некоторые из них будут рассмотрены позже.

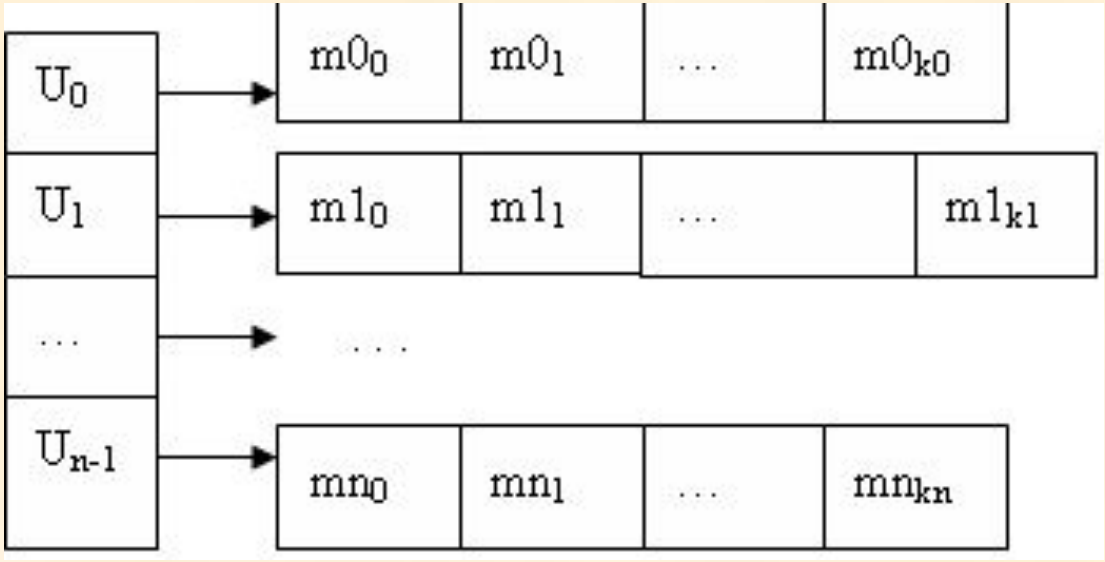
Свободные массивы

Свободными называют двухмерные массивы, размер каждой из строк которых может быть различным.
Поскольку стандарты языка программирования не допускают такого вида массивов, то создается симбиоз массивов:

- одномерный массив указателей, число элементов ***n*** которого равно числу строк переменной длины,
- и ***n*** одномерных массивов различной длины.

Таким образом, массив указателей имеет фиксированную длину, значит, фиксировано и число строк свободного массива. Память под каждую строку свободного массива запрашивается динамически.
Следовательно, при создании такого массива требуется (***n***+1) обращений к ОС для получения динамической памяти. После того как надобность в свободном массиве отпала, необходимо освободить занимаемую им память, что опять-таки потребует (***n***+1) обращений к ОС.

Структура хранения свободного массива с ***n*** строками:



Массив указателей, ***U*** имеет ***n*** элементов, каждый из которых содержит адреса векторов-строк массива.

Для управления свободным массивом может создаваться ***дескриптор***, в простейшем случае - это указатель на массив указателей.

Свободные массивы используются для представления других, более сложных структур. С примерами таких массивов мы встретимся ниже.

По аналогичной схеме могут быть созданы и двухмерные массивы с одинаковыми длинами строк.

К ***недостаткам*** можно отнести потребность в дополнительной памяти под массив указателей и многократное обращение к ОС для получения памяти.

К ***преимуществам*** можно отнести следующее:

1) при работе с очень большими массивами получение больших сплошных областей может стать невозможным; в то же время получение памяти под каждую строку вполне возможно;

2) появляется возможность обрабатывать большие массивы данных, хранящихся в файлах, размещая в оперативной памяти только те строки, которые совместно обрабатываются в данный момент;

3) применение свободных массивов позволяет более эффективно использовать память;

4) при сортировке массива по строкам (не элементов в строке, а самих строк) вместо перестановок строк в некоторых случаях можно переставлять только элементы массива указателей, в то время как сами строки остаются на месте.

Треугольные и разреженные матрицы

Иногда при использовании матриц имеет смысл хранить только какую-либо часть матрицы. К таким матрицам можно отнести треугольные и разреженные матрицы.

В ***треугольной матрице*** A_{n*n} все элементы выше или ниже главной диагонали являются нулевыми.
Если нулевыми являются элементы выше диагонали (***нижняя треугольная матрица***), то ненулевыми будут элементы

$$a_{11}; a_{21}, a_{22}; a_{31}, a_{32n}; a_{33}; \ldots; a_{n1}, a_{n2}, \ldots, a_{nn}.$$

Нижнетреугольную матрицу целесообразно хранить в виде одномерного массива (вектора) ***B*** с числом элементов ***m=n*(n+1)/2***.

Тогда индекс ***ib*** в массиве ***B*** для исходного элемента a_{ij}, $i=1, 2, \ldots, n$, $j=1, 2, \ldots, i$, определяется по формуле

$$ib = \sum_{k=1}^{i-1} k + j = (i^2 - i)/2 + j; ib = 1, 2, \ldots, n(n+1)/2.$$

Если индексация массивов начинается с нуля, то индексы элемента a_{ij}, $i=0, 1, \ldots, n-1$, $j=0, 1, \ldots, i$, вычисляются по формуле

$$ib = \sum_{k=1}^{i} k + j = (i^2 - i)/2 + j; ib = 0, 1, 2, \ldots, n(n^2 + n)/2 - 1.$$

Если в исходной матрице нулевыми являются элементы, расположенные ниже главной диагонали (***верхняя треугольная матрица***), то в массиве ***B*** ненулевые элементы разместятся в последовательности

$$a_{11}, a_{12}, \ldots, a_{1n}; a_{22}, a_{23}, \ldots, a_{2n}; \ldots; a_{nn},$$

а доступ к элементу a_{ij} в этом случае будет осуществляться по индексу ***ib***, вычисляемому по более сложной формуле

$$ib = n(i-1) + j - \sum_{k=1}^{i-1} k = n(i-1) + j - (i^2 - i)/2; ib = 1,2,\ldots, n(n+1)/2.$$

При индексации, начинающейся с нуля, формула преобразуется

$$ib = n*i + j - \sum_{k=1}^{i} k = n*i + j - (i^2 + i)/2; ib = 0,1,2,\ldots, n(n+1)/2 - 1.$$

Разреженными называются матрицы, содержащие большое количество нулевых элементов.
Они обычно встречаются в научных приложениях, при представлении графов в программах.
Представление таких матриц в виде двухмерных массивов приводит к нерациональному использованию памяти и неэффективности выполняемых над ними операций.
Существуют различные способы представления разреженных матриц. Все они сводятся к сохранению только ненулевых элементов матрицы и их индексов.

Первый и наиболее простой способ заключается в следующем.

- Создается двухмерный массив (***n****3), где ***n*** - число ненулевых элементов исходной матрицы. Первые два элемента каждой строки содержат индексы строки и столбца ненулевого элемента, а третий - сам ненулевой элемент.
- Если же значения элементов исходной матрицы нецелого типа, то вместо одного двухмерного массива (***n****3) создаются три одномерных массива с ***n*** элементами каждый.
- Обработка таких массивов особых затруднений не вызывает.

 Второй способ. Имеется возможность несколько сократить объем требуемой памяти.

- Создаются три массива: массив ***STR*** по числу строк исходной матрицы; массивы ***STL*** и ***ZN*** по числу ненулевых элементов исходной разреженной матрицы.
- В массив ***STL*** последовательно заносятся индексы столбцов ненулевых элементов исходной матрицы, сначала для первой строки, затем для второй и т.д., т.е. индексы столбцов каждой ***i***-и строки образуют ***i***-ю группу.
- Каждый ***i***-й элемент массива ***STR*** содержит индекс начала ***i***-й группы в массиве ***STL***.
- В массив ***ZN*** заносятся значения соответствующих ненулевых элементов.
- Алгоритмы обработки разреженной матрицы при таком представлении несколько усложняются, так как число ненулевых элементов в строках исходной матрицы различно, а массив ***STL*** не содержит признаков конца групп. Поэтому для перехода с одной строки матрицы на другую нужно использовать информацию из массива ***STR***.

 Третий способ. Рассмотренные способы представления разреженных матриц массивами удобны только тогда, когда число и расположение ненулевых элементов остаются неизменными. В противном случае более подходящим является способ представления разреженной матрицы с использованием многосвязных циклических списков с динамическим получением памяти для элементов списка.

Элемент структуры списка

Указатель на следующий элемент в списке строки		Указатель на следующий элемент в списке столбца
Номер строки	Номер столбца	Значение элемента

- Для каждой строки и каждого столбца строятся циклические списки. Ненулевой элемент a_{ij} исходной матрицы попадает в два списка - ***i***-й циклический список для ***i***-й строки и ***j***-й циклический список для ***j***-го столбца. Поэтому каждый элемент структуры должен содержать два указателя - указатель на следующий элемент в списке строки и указатель на следующий элемент в списке столбца.
- Каждый список начинается с головного элемента. В головном элементе списка используется только одно поле - указатель по строке в списке строки или указатель по столбцу в списке столбца. Элемент списка, представляющий ненулевой элемент матрицы, использует все пять полей структуры элемента.

3.2. Записи

Запись представляет собой совокупность ограниченного числа логически связанных компонент, принадлежащих к разным типам.

Компоненты записи называются ***полями***, каждое из которых определяется именем. Сами поля, в свою очередь, могут быть составными.

В математике такие составные типы называют декартовым произведением составляющих его типов.

Мощность составляющего типа есть произведение мощностей составляющих его типов.

Каждый элемент в записи имеет свое уникальное имя, но такое же имя может использоваться в других записях или для обозначения других объектов программы.

Записи, как и массивы, состоят из фиксированного числа элементов, но между массивами и записями имеются два существенных различия.

❶***Во-первых***, в отличие от массива, состоящего из однотипных элементов, элементы записи могут быть разных типов.

❷***Во-вторых***, в то время как доступ к элементам массива осуществляется посредством индексов, доступ к элементам записи - по их именам. Отдельные поля (элементы) записи могут служить в качестве ключей записей.

Записи, как правило, используются в других, более сложных структурах:

- ♦ в таблицах,
- ♦ файлах,
- ♦ базах данных.

Отдельная запись используется редко, обычно для извлечения и обработки элемента из более сложной структуры.

Примером записи может служить запись с данными о служащем:

- ♦фамилия;
- ♦имя;
- ♦отчество;
- ♦дата рождения;
- ♦дата поступления на работу;
- ♦специальность;
- ♦семейное положение.

Элементы записи «дата рождения» и «дата поступления на работу», в свою очередь, могут рассматриваться также как запись:

- ♦день;
- ♦месяц:
- ♦год.

Запись хранится в одной сплошной области памяти, причем, ее элементы размещаются в памяти последовательно друг за другом в том порядке, в котором они перечислены в записи, например: фамилия, имя, отчество, день, месяц и год рождения, день, месяц и год поступления на работу, специальность, семейное положение.

Операции над записями

Важнейшей операцией для записи является операция доступа к выбранному полю записи - ***операция квалификации***.

Практически во всех языках программирования обозначение этой операции имеет вид:

< имя переменной-записи >.< имя поля >

Над выбранным полем записи возможны любые операции, допустимые для типа этого поля.

Большинство языков программирования поддерживает некоторые операции, работающие с записью, как с единым целым, а не с отдельными ее полями.

Это ***операции присваивания*** одной записи значения другой однотипной записи и ***сравнения*** двух однотипных записей на равенство/неравенство.

В тех же случаях, когда такие операции не поддерживаются языком явно (язык С), они могут выполняться над отдельными полями записей или же записи могут копироваться и сравниваться как неструктурированные области памяти.

3.3. Множества

С термином ***множество*** в математике и в языках программирования, при манипулировании со структурами данных связывается разный смысл.

В **математике** ***множество*** это любая совокупность объектов, выбранная из универсального множества. Универсальным при этом считается множество, содержащее сразу все рассматриваемые элементы.

Элементами множества в математике могут быть данные различных типов, число элементов не ограничено, одинаковых элементов нет.

Множество определяется перечислением свих элементов как $A=\{a_1, a_2, \ldots, a_n\}$, a_i – элементы множества.

Если x элемент множества A, то записывают $x \in A$.

Возможно другое определение множества через характеристическое свойство своего элемента:

$A=\{x \mid x$ – день недели$\}$.

Если A подмножество B, то пишут $A \subseteq B$, пустое множество обозначают $A = \varnothing$ или $A=\{\ \}$.

Операции над множествами (математика)

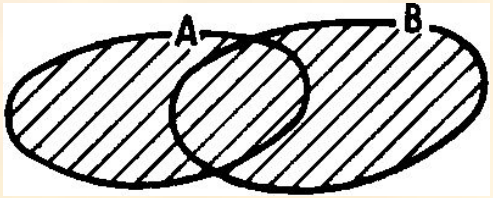
Пусть ***A*** и ***B*** некоторые множества элементов из некоторого класса объектов ***U***, тогда определены следующие операции.

1. **Дополнение - унарная операция**

A= { ***x***| ***x*∉*A*** } содержит все те элементы ***U***, которые не являются элементами множества ***A***.

2. **Объединение множеств *A* и *B***

A*∪*B={ ***x***| ***x*∈*A*** или ***x*∈*B***} содержит все элементы ***U***, каждый из которых является либо элементом множества ***A***, либо элементом множества ***B***, либо одновременно элементом множества ***A*** и элементом множества ***B***.



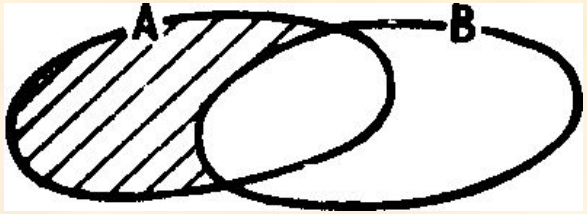
Операции над множествами (математика)

3. **Пересечение множеств *A* и *B***

A*∩*B={ ***x***| ***x*** ∈ ***A*** и ***x*** ∈ ***B***} содержит все элементы ***U***, каждый из которых является одновременно элементом множества ***A*** и элементом множества ***B***.

4. **Вычитание множеств *A* и *B***

A-B={ ***x***| ***x*** ∈ ***A***, но ***x***∉***B***} содержит все те элементы ***U***, каждый из которых является элементом множества ***A***, но не является элементом множества ***B***.



Операции над множествами (математика)

5. **Произведение множеств *A и B*** называется такое множество, каждый элемент которого представляет собой совокупность двух объектов (пару), при этом один из объектов пары является элементом из ***A***, а второй – элементом из ***B***:

$$AxB=\{ (a, b)|\ a \in A, \text{ но } b \in B\}.$$

Любое подмножество множества ***AxB*** есть отношение ***R***, при этом множество ***A*** называется областью определения, а множество ***B*** – областью значений. Отношение ***R*** часто имеет смысл =, >, <, и т.д.

6. **Функция (отношение, преобразование) f**: $A \cap B$ или $f: A \rightarrow B$ есть множество пар элементов (***a, b***), таких, что $a \in A$, $b \in B$ и $b=f(a)$.

В **языках программирования** , например в Паскале, предусмотрены структуры типа «множество» (множественные типы).

Множество – это структурированный тип данных, представляющий собой набор взаимосвязанных по какому-либо признаку или группе признаков неповторяющихся объектов, которые можно рассматривать как единое целое.

Каждый объект в множестве называется ***элементом множества***.

Все элементы множества должны принадлежать одному из скалярных типов, кроме вещественного. Этот тип называется ***базовым типом*** множества. Базовый тип задается диапазоном или перечислением.

Размер множества ограничен некоторым предельно допустимым количеством элементов, например, в Паскале это 256, значения элементов могут изменяться только в пределах от нуля до 255.

Поэтому базовым типом множества могут быть ***byte***, ***char*** и производные от них типы.

Множество в памяти хранится как массив битов, в котором каждый бит указывает является ли элемент принадлежащим объявленному множеству или нет. Т.о. максимальное число элементов множества 256, а данные типа множество могут занимать не более 32-ух байт.

Число байтов, выделяемых для данных типа множество, вычисляется по формуле:

ByteSize = (max div 8)-(min div 8) + 1,

где **max** и **min** - верхняя и нижняя границы базового типа данного множества.

Номер байта для конкретного элемента **E** вычисляется по формуле:

ByteNumber = (E div 8)-(min div 8),

номер бита внутри этого байта по формуле:

BitNumber = E mod 8

Стандарт языка определяет операции над множествами, таковыми в Паскале являются:

- ♦объединение (+),
- ♦пересечение (*),
- ♦разность (-),
- ♦проверка принадлежности элемента множеству (in).

Предусмотрены также операции сравнения множеств =, <>, <=, >=, например, A<=B - операция проверки того, является ли A подмножеством B.

В языке Си структура типа «множество» не предусмотрена.

Множество как обобщенное понятие структур данных

С точки зрения **структур данных** ***множество*** можно рассматривать как совокупность данных, над которыми выполняется некоторое число операций, образующих функциональную спецификацию структуры этого множества.

Пусть определен некоторый тип данных ***T***. Определим другой тип, элементами которого является множество объектов типа ***T***.

Над данными этого множественного типа допустимы следующие основные операции:

- ***создать множество*** – создаётся пустое множество, возвращается его адрес;
- ***включить элемент*** – формируемся новое множество добавлением одного элемента к существующему множеству;
- ***найти элемент*** – проверить, есть ли элемент в множестве, если есть, определить его адрес;
- ***удалить элемент*** – формируется новое множество с удалением одного элемента, если он есть; в противном случае множество остается без изменения;
- ***пусто*** – проверить, есть ли элементы во множестве;

- ***выборка*** – выдается элемент для обработки, если он есть; в простейшем случае определяется только его адрес, по которому осуществляется обработка;
- ***выборка с удалением*** – элемент выбирается для обработки, затем удаляется из множества;
- ***объединение множеств*** – над множествами выполняются операции как над математическими множествами, возможны некоторые модификации таких операций.

Многоэлементные структуры, которые мы будем рассматривать, такие, как стеки, очереди, деревья, таблицы, представляют собой частные случаи понятия «множество», а перечисленные выше операции над различными структурами могут иметь другие названия и различные алгоритмы их реализации.
Эти алгоритмы в существенной мере зависят от физической структуры представления данных, составляющих множества.