1. (стр. 66) Понятие информации, виды и способы её представления

Термин информация происходит от латинского informatio, что означает «сведения, разъяснения, изложение».

Информация - это отображение окружающего нас мира с помощью знаков и сигналов или иначе сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые воспринимают информационные системы (живые организмы, управляющие машины и др.) в процессе жизнедеятельности и работы.

Информация передается от источника к приемнику в материально-энергетической форме в виде сигналов (например: электрических, световых, звуковых и т.д.), распространяющихся в определенной среде.

(сигнал – это физический процесс (явление), несущий сообщение (информацию) о событии или состоянии объекта наблюдения)

Понятие ИНФОРМАЦИЯ предполагает наличие:

* материального носителя информации
* источника информации
* передатчика информации
* приемника
* канала связи между источником и приемником.

Существует множество способов передачи и обработки информации. Человек может передавать информацию, используя тот или иной язык, жесты, мимику, звуки и воспринимать информацию используя любые органы чувств. Иными словами, информация человеком передается, обрабатывается и принимается в форме знаков или сигналами. Сигнал может быть световым, звуковым (радиоволны), электромагнитным, биохимическим и т.д.

Знаками можно считать алфавит любого языка, знаки языка жестов, любые коды или шифры, нотные значки и т.д.

Информацию можно: создавать, принимать, комбинировать, хранить, передавать, копировать, обрабатывать, и тд. Все эти процессы, связанные с определенными операциями над информацией, называются информационными процессами.

Основные виды информации по ее форме представления, способам кодирования и хранения:

* графическая или изобразительная

первый вид – наскальные рисунки, позднее картины, фотографии

* звуковая
* текстовая
* числовая
* видеоинформация

Символьная информация может представляться с использованием самых разных алфавитов (набором символов) без искажения и смысла информации: при необходимости можно изменять форму представления информации – вместо общепринятого алфавита использовать другой, например, двухбуквенный.

Форма представления информации, отличная от естественной, общепринятой, называется кодом. Почтовые индексы, телеграфный код Морзе – всё это коды.

1. (стр. 69) Задачи получения, передачи, преобразования и хранения информации

Услышав (или увидев) фразу, произнесенную (или написанную) на языке, которого человек не знает, он получает информацию, но не может определить смысл. Поэтому для передачи смыслового содержания информации необходимы некоторые соглашения между источником и приемником о смысловом содержании сигналов. Такие соглашения могут быть достигнуты в процессе общения.

Обработка информации — получение одних информационных объектов из других путем выполнения некоторых алгоритмов.

Обработка является одной из основных операций, выполняемых над информацией, и главным средством увеличения объема и разнообразия информации.

Средства обработки информации — это всевозможные устройства и системы, созданные человечеством, и в первую очередь компьютер — универсальная машина для обработки информации. Компьютеры обрабатывают информацию путем выполнения некоторых алгоритмов. Живые организмы и растения обрабатывают информацию с помощью своих органов и систем.

Информационные ресурсы — это идеи человечества и указания по реализации этих идей, накопленные в форме, позволяющей их воспроизводство. Информационные ресурсы (в отличие от всех видов ресурсов — трудовых, энергетических, минеральных) тем быстрее растут, чем больше их расходуют.

Информационные технологии — это совокупность методов и устройств, используемых людьми для обработки информации.

Хранение информации осуществляется с помощью её переноса на некоторые материальные носители. В настоящее время особое значение получило хранение информации в виде последовательностей двоичных символов. Для реализации этих методов используются разнообразные запоминающие устройства.

Хранить информацию можно:

* в форме знаков на бумаге, перфокартах, перфоленте, на камне, дереве, ткани, в рельефе ключа и т.п.;
* в форме электромагнитных сигналов на дисках, дискетах, кассетах, кинопленке и т.п.;
* в форме структурных комбинаций в клетках и генах биологических объектов;

Передачей семантической информации называется процесс ее пространственного переноса от источника к получателю.

Для представления информации используются различные знаковые системы – наборы заранее оговоренных смысловых символов: предметов, картинок, написанных или напечатанных слов естественного языка.

Передача информации происходит:

* в форме сигналов, как в технических устройствах, так и в общении людей, в телекоммуникациях, в жизни общества, в живых существах;
* механическим путем в общении людей и в технических устройствах.

К техническим средствам хранения, передачи и обработки информации относятся:

* различные диски, дискеты,
* аудио и видео кассеты,
* объекты телекоммуникации (радио, телевидение, телефон, телеграф, персональные компьютеры и компьютерные сети и т.п.)

ЭВМ — это универсальная вычислительная машина, которая обрабатывает информацию, используя электрические сигналы. Если сигнал непрерывен во времени, то его называют аналоговым, и машины, использующие такие сигналы, называют аналоговыми.

Если сигнал имеет прерывистую (дискретную) форму, он называется цифровым (дискретным), и машины соответственно называются цифровыми. Современные ПК относятся к ЦВМ.

Всю информацию ПК может воспринимать в форме сигналов, которые имеют только два различных состояния: есть сигнал или нет сигнала. Этим сигналам поставлен в соответствие разряд кода, который может принимать только два значения: 0 или 1. Сигнал, который имеет только два различных значения, или соответствующий ему разряд кода, который может принимать только два значения - 0 или 1, называется бит. Бит — это минимальная единица информации.

На практике чаще применяется более крупная единица — байт, равная восьми битам. Именно восемь бит требуется для того, чтобы закодировать любой из 256 символов (256 = 28).

1. (стр. 72) Язык как способ представления и передачи информации

Знаковая форма восприятия, хранения и передачи информации означает использование какого-либо языка. Языки делятся на:

* разговорные (естественные).

Естественные языки носят национальный характер.

* формальные.

Чаще всего относятся к специальной области человеческой деятельности (например, язык математики или язык флажков на флоте).

Информация во внешней среде выражается с помощью некоторых материальных объектов (носителей), ассортимент и способ расположения которых задает информацию.

Человек воспринимает сообщение посредством органов чувств. Приемник информации в технике воспринимает сообщения с помощью различной измерительной и регистрирующей аппаратуры. Носителем информации в различных информационных процессах может быть, например, камень, бумага, электрический кабель, магнитный диск.

Отображение множества состояний источника во множество состояний носителя называется способом кодирования. Таким образом, при выбранном способе кодирования какое-либо состояние заменяется своим образом — кодом состояния (или кодом информации, задаваемой этим состоянием).

Итак, последовательность символов алфавита, кодирующая состояние источника и воспринимаемая адресатом как сообщение, как информация, образует слово на этом языке. На передачу и переработку информации влияет то, сигналами какой природы отображается одна и та же информация, то есть каким кодом задана одна и та же информация.

Если говорить о сигналах, дискретных по виду, то их множество конечно, поэтому их принято кодировать буквами алфавита того или иного естественного языка или цифрами той или иной системы счисления.

Таким образом, дискретная информация отождествляется с алфавитно-цифровой, а простейшим алфавитом, достаточным для записи (представления) информации, является алфавит из двух символов, допустим 0 и 1.

Для обмена информацией с другими людьми человек использует естественные языки, в основе которых лежит алфавит, то есть набор символов (знаков), которые человек различает по их начертанию.

Наряду с естественными языками были разработаны формальные языки (системы счисления, язык алгебры, языки программирования и др.).

1. Задачи, требующие автоматизированной обработки информации
2. Системы автоматизированной и автоматической обработки информации
3. Исторические этапы автоматизации обработки информации
4. Последовательность обработки прикладных программ
5. Жизненный цикл программного обеспечения
6. Задачи системного программного обеспечения
7. Назначение и структура операционных систем
8. Назначение и последовательность функционирования компиляторов
9. (стр. 74) Требования к языкам программирования и их классификация

Языки программирования – языки для записи программ для компьютеров. Это совокупность средств и правил представления алгоритма в виде, приемлемом для компьютера. Существуют различные классификации языков программирования. По наиболее распространённой классификации все языки программирования делят на языки низкого и высокого уровня. Если язык близок к естественному языку программирования – это язык высокого уровня, если ближе к машинным командам – язык низкого уровня.

1. Понятие алгоритма, его основные свойства и способы записи
2. Временная и объёмная сложность алгоритма
3. Оценки временной сложности алгоритма
4. Структура рекурсивных алгоритмов
5. Структура программы на языке С++
6. Методы доступа к функциям из библиотек и пакетов
7. Описания, выражения и операторы в языке С++
8. Арифметические типы данных в языке С++
9. Символьные и логические данные в языке С++
10. Операции над данными стандартных типов и их старшинство
11. Потоковый ввод и вывод информации в языке С++
12. Манипуляторы потокового ввода и вывода
13. Ссылочные типы данных и их применение
14. Указатели в языке С++. Доступ к данным по указателю.

Указатель – это переменная, значением которой является адрес памяти, по которому хранится объект определенного типа (другая переменная).

а) Значение переменной указателя – адрес другой переменной.

б) Андрес переменной указателя свой и неизменный.

Например, если С – это переменная типа char, а Р – указатель на С, значит в Р находится адрес, по которому в памяти компьютера хранится значение переменной С.

Как и любая переменная, указатель должен быть объявлен. При объявлении указателей всегда указывается тип объекта, который будет хранится по данному адресу. Принцип объявление указателей такой же, как и принцип объявления переменных, но у указателя перед именем ставится символ звездочки \*. Визуально указатели отличаются только этим символом. При объявлении указателей компилятор выделяет несколько байт памяти, в зависимости от типа данных, отводимых для хранения некоторой информации в памяти.

Чтобы получить значение, записанное в некоторой области, на которое ссылается указатель нужно воспользоваться операцией разыменования указателя\*. Необходимо поставить звездочку перед именем и получим доступ к значению указателя.

Указатель можно создать на любой тип, как стандартный, так и определённый пользователем за исключением ссылочных. В С++ различают указатели на объект и на функцию.

Указатель на объект содержит адрес области памяти, в которой хранятся данные определённого типа, в том числе возможен указатель на указатель <тип>\*<имя\_указателя> = &<имя>;.

Среди указателей на объект можно выделить указатель на тип void. Он применяется в тех случаях, когда конкретный тип объекта, на которой находится указатель, не определен. Перед выполнением каких-либо действий с областью памяти, на которую он указывает, требуется преобразовать его к конкретному типу.

Указатель на функцию содержит адрес, по которому передается управление при вызове функции.

1. Структурный тип данных
2. Массивы в языке С++. Связь массивов и указателей

Массив – это область памяти, где могут последовательно храниться несколько значений. Массив состоит из конечной последовательности элементов одного и того же типа.

Для объявления массива следует указать тип, имя массива, а также количество значений, хранящихся в массиве. Значения внутри массивы – элементы этого массива. Допускается создание многомерных массивов как массивов, элементами которых являются массивы.

Каждый элемент массива имеет индекс (номер), с его помощью можно определить любой элемент. Индексы всегда начинаются с 0. А индекс последнего элемента равен длине массива минус 1.

Можно объявить массив, не указываю явно, сколько в нем элементов.

int mas [] = {3, 4, 19, 2} // выделяется 4 байта

Символьный массив отличается от остальных тем, что после символов стоит значок \0 – символ, означающий, что перечисление символов закончено.

char mas [] = {‘3’, ‘4’, ‘19’, ‘2’} // выделяется 5 байт, так как ‘\0’- 1 символ.

1. Многомерные массивы

Многомерные массивы – это массивы, у которых есть более одного индекса. Вместо одной строки элементов, многомерные массивы можно рассматривать как совокупность элементов, которые распределены по двум или более измерениям.

В C/C++ прямоугольный двумерный массив чисел действительно реализует математическое понятие «матрица». Однако, в общем случае, *двумерный массив* — понятие гораздо более широкое, чем матрица, поскольку он может быть и не прямоугольным, и не числовым.

Определение многомерного массива в общем случае должно содержать сведения о типе, размерности и количествах элементов каждой размерности, например: int mas[4][3] (массив, имеющий 4 строки и 3 столбца).

void input (int \*\*mas, int &n){ //заполнение двумерного динамического массива

for (int i=0; i<n; ++i) {

for (int j=0; j<n; ++j) {

mas[i][j] = rand () % 100 - 50;}

}

}

int main(){

int n;

cin >> n;

int \*\*mas = new int\* [n];

for(int i = 0; i < n; ++i){

mas[i] = new int[n];}

input(mas,n);

1. Строки и массивы символов

Символ – элементарная единица, некоторый набор которых несет определённый смысл. В языке программирования С++ предусмотрено использование символьных констант.

Символьная константа – это целочисленное значение (типа int) представленное в виде символа, заключенного в одинарные кавычки, например ‘a’.

Нуль-символ, в языке С++ он используется для того, чтобы отмечать конец строки, Нуль-символ – не есть цифра ноль; он не выводится на печать и в таблице кодом аски имеет номер 0. Наличие нуль-символа означает, что количество ячеек массива должно быть по крайней мере на одну больше, чем количество символов, которые необходимо размещать в памяти.

char mas[5] = {‘a’, ‘b’, ‘c’, ‘d’} // один символ под \0

Для символьных массивов существует специальный способ инициализации. Вместо фигурных скобок и запятых можно прямо использовать строку символов, заключенных в кавычки. При этом в описании не обязательно задавать размеры массива, по сколько компилятор “сам” определяет его длину, подсчитывая число начальных значений.

char mas[] = “Home!”; //инициализация символьного массива.

Если программа использует строковые константы, заключенные в двойные кавычки, компилятор С++ автоматически добавляет символ NULL. С++ позволяет инициализировать строки при объявлении, указывая требуемые символы внутри двойных кавычек.

1. Файлы и потоки

Файлом – называют последовательность байтов, хранящихся на внешнем носителе информации. Под доступом к файлу понимают запись и чтение данных из файла.

Потоком – называется логический интерфейс (программа), который обеспечивает доступ к файлу. Каждый поток может работать в двух режимах: текстовом и бинарном. Режим работы потока задается при его соединении с файлом.

Прежде чем использовать поток для доступа к файлу, его необходимо соединить с этим файлом, то есть обеспечить поток информацией о файле. Эта информация хранится в структуре типа FILE. Потому считается, что поток имеет тип FILE\*, то есть является указателем на файл. Когда поток соединяют с файлом, то говорят, что файл открывают. Когда поток отсоединяют от файла, то говорят, что файл закрывают.

В текстовом режиме поток записывает и читает из файла текстовые строки, которые заканчиваются символом ‘\n’ и могут содержать символ ‘\t’. По стандарту поток должен обеспечивать обработку строк длиной не менее 254 символа, включая символ ‘\n’. Стандартом допускается, что при чтении и записи данных текстовым потоком может происходить их преобразование.

В бинарном режим поток записывает и читает данные из файла в том виде, в котором они хранятся в оперативной памяти.

ifsteram - для ввода из файла ;

ofsteram - для вывода в файл;

fsteram - для обмена с файлом в двух направлениях.

1. Методы открытия файлов для обмена информацией

При операциях с файлом вначале необходимо открыть файл с помощью функции open().

Для открытия файла в функцию необходимо передать путь к файлу в виде строки. И также можно указать режим открытия. Режимы открытия файлов устанавливают характер использования файлов.

* ios::in: файл открывается для ввода (чтения). Может быть установлен только для объекта ifstream или fstream
* ios::out: файл открывается для вывода (записи). При этом старые данные удаляются. Может быть установлен только для объекта ofstream или fstream
* ios::binary: файл открывается в бинарном режиме
* ios::app: файл открывается для добавления информации в конец файла

ofstream fout("text.txt", ios::app); // открываем файл для добавления информации к концу файла.

fout.open("text.txt", ios::app); // открываем файл для добавления информации к концу файл.

Если при открытии режим не указан, то по умолчанию для объектов ofstream применяется режим ios::out, а для объектов ifstream - режим ios::in. Для объектов fstream совмещаются режимы ios::out и ios::in.

После завершения работы с файлом его следует закрыть с помощью функции close(). Также стоит отметить, то при выходе объекта потока из области видимости, он удаляется, и у него автоматически вызывается функция close.

1. Прямой доступ к отдельным записям в файлах
2. Программная реализация алгоритмов линейной структуры
3. Программная реализация разветвляющихся алгоритмов
4. (стр. 36) Реализация циклических алгоритмов с заранее неизвестным количеством повторений

Циклы используются для многократного повторения кусков кода.

Для программирования циклических алгоритмов с заранее неизвестным количеством повторений в С++ используется два оператора: “повторять пока выполняется условие” (while), “выполнять до выполнения условия” (do … while).

Цикл while

Используется, если выход из цикла должен осуществляться при нарушении некоторого условия, заданного в виде выражения. While (<выражение>) <оператор>;

Тело цикла выполняется до тех пор, пока результат выражения отличен от нуля (true).

Цикл do … while

Используется, если выход из цикла также при нарушении некоторого условия, но проверка должна производиться после выполнения текла цикла. Do <оператор> while (<выражение>);

Тело цикла выполняется, пока результат выражения отличен от нуля (true). Но в отличие от цикла while, тело цикла всегда выполняется хотя бы один раз.

Операторы break и continue

Эти два ключевых слова очень важны для циклов, они могут управлять повторениями в цикле.

Оператор break делает принудительный выход из цикла, даже когда условие цикла – истинно. Оператор break удобно использовать, когда необходимо выйти из цикла при особых обстоятельствах.

Оператор continue нужен если необходимо пропустить блок кода, но при этом не прекращать работу цикла.

1. (стр. 37) Реализация циклических алгоритмов с заранее известным количеством повторений

Цикл for

Оператор for это наиболее общий способ организации цикла. Имеет формат:

For (выражение 1; выражение 2; выражение 3) <тело>

* выражение 1: обычно используется для установления начального значения переменных, управляющих циклом
* выражение 2: выражение, определяющие условие, при котором тело цикла будет выполняться.
* выражение 3: определяет изменение переменных, управляющих циклом после каждого выполнения тела цикла. В качестве тела может быть использован любой оператор, в том числе и пустой или составной.

Любой из выражений, а также все сразу, могут быть опущены, при этом разделяющие их символы ‘; ’ пропускать нельзя.

Схема выполнения:

1. вычисляется выражение 1.
2. вычисляется выражение 2.
3. Если значения выражения\_2 отлично от нуля (истино), выполняется тело цикла, вычисляется выражение\_3 и осуществляется переход к пункту 2,

а если выражение\_2 равно нуль(ложь), выполнение оператора for завершается и управление передается на оператор, следующий за оператором for. При отсутствии выражения\_2 оно подразумевается истинным.

Существенно то, что проверка условия всегда выполняется в начале цикла. Это значит, что тело цикла не может ни разу не выполниться, если условие выполнения сразу будет ложным.

Другим вариантом использования оператора for является бесконечный цикл. Для организации такого цикла можно использовать пустой условное выражение, а для выхода из цикла обычно используют условный оператор и оператор break.

for (; ;) {

If (некоторое условие) break;}

Согласно синтаксису языка Си тело цикла должно иметься у любого цикла, но если по смыслу оно не требуется, то в качестве тела можно использовать оператор. Такой цикл может быть использован, например, для организации поиска.

Оператор break делает принудительный выход из цикла, даже когда условие цикла – истинно. Оператор break удобно использовать, когда необходимо выйти из цикла при особых обстоятельствах.

Оператор continue нужен если необходимо пропустить блок кода, но при этом не прекращать работу цикла.

1. (стр. 39) Структура функции в языке С++. Прототип, заголовок и вызов функции.

Функция – один из самых важных компонентов языка С++.

* Любая функция, также, как и любая переменная, имеет тип.
* Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.
* Если функция не возвращает никакого значения, то она должна иметь тип void (такие функции иногда называют процедурами)
* При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки – открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может вообще не быть
* После списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции
* В конце тела функции обязательно ставится закрывающая фигурная скобка.

Прототип – это явное объявление функции, которое предшествует определению функции.

Прототип функции – это заголовок функции, который объявляется перед функцией main(). И если объявить прототип функции, тогда функцию можно будет увидеть в main ().

Тип возвращаемого значения при объявлении функции должен соответствовать типу возвращаемого значения в определении функции.

Если функция определяется до того, как она вызывается, то определение функции служит одновременно ее прототипом, так что отдельный прототип в этом случае не является необходимым. Если функция вызывается до ее определения и прототип для нее отсутствует, происходит ошибка компиляции.

Наличие в прототипе полного списка типов аргументов параметров позволяет выполнить проверку соответствия типов фактических параметров при выходе функции типам формальных параметров, и, если необходимо, выполнить соответствующие преобразования.

---------

1. (стр. 41) Предварительное описание функции

Предварительное объявление является таким типом объявления, при котором компилятор имеет возможность разрешить ссылки из различных частей программы. Предварительное объявление позволяет программисту ссылаться на объекты, о которых компилятор ещё не знает, но которые будут определены в процессе компиляции позже.

Предварительное объявление – объявление идентификатора (типа, переменной или функции), для которого программист ещё не дал полного определения. Объявление идентификатора требуется компилятору для того, чтобы знать тип(размер) идентификатора, но не его значения (в случае переменных).

Предварительное описание функции задает ее имя, тип возвращаемого значения и список типов, передаваемых параметров. Список прототипов всех функций, как правило, размещается перед определением первой функции и необходим при вызове функциями друг друга, то есть функция может вызвать функцию, описание которой расположено после этой функции.

При записи функции в библиотечные файлы, прототипы отправляются в файлы с расширением \*.h.

Объявления функций также называют прототипами функций, поскольку они являются неким общим представлением или описанием функций.

Определения функций всегда появляются в программном коде раньше, чем к ним выполняется обращение. Большинство современных С-компилятором требует, чтобы информация о типе возвращаемых значений и тапах параметром функций была известна до вызова этих функций в программе. Расположение определения функции ранее первого ее использования обеспечивает компилятор требуемой информацией. Однако, по мере усложнения программы, размещение функций в исходном коде с соблюдением правильного порядка может оказаться невозможным. В таких случаях следует использовать прототипы, описывающие возвращаемые значения и типы параметров функций.

1. (стр. 42) Локальные и глобальные величины программы

Описание устанавливают имена и типы именованных констант, переменных, функций, а также имена типов, созданных пользователем.

* Глобальные величины описываются вне оперативных скобок любых функций и действуют от конца описания до конца файла.
* Локальные величины описываются внутри блока и действуют от конца описания до конца блока.

Локальная переменная – это переменная, которую можно объявлять внутри функций. Такие переменные называются локальными, поскольку их имена и значения имеют смысл только внутри функций, в которых они определены.

Локализация переменных дает программисту большую свободу в выборе идентификаторов. Так, если две процедуры a и b полностью отделены друг от друга (т.е. не вложены одна в другую), то идентификаторы в них могут быть выбраны совершенно произвольно, в частности, могут повторяться. В этом случае совпадающим идентификаторам соответствуют разные области памяти, совершенно друг с другом не связанные.

Глобальная переменная – это переменная, которой могут оперировать все функции программы. В действительности, в зависимости от выбора места определения глобальной переменной, можно контролировать набор функций, имеющих доступ к этой переменной. Другими словами, можно управлять областью действия глобальных переменных. Функции, определения которых располагаются до определения глобальной переменной, не имеют доступа к этой переменной.

К сожалению, использование глобальных переменных часто становится причиной появления большого количества ошибок. Поскольку значение глобальной переменной может быть изменено фактически из любого места программы, это очень усложняет поиск в программе всех мест изменения этих переменных.

1. (стр. 43) Способы обмена информацией между вызывающей и вызываемой функциями

Механизм параметров является основным способом обмена информацией между вызываемой и вызывающей функциями.

Параметры, перечисленные в заголовке описания функции, называются формальными, а записанные при вызове функции – фактическими (или аргументами).

При вызове функции:

* в первую очередь вычисляются выражения, стоящие на месте фактических параметров;
* затем выделяется память под формальные параметры функции в соответствии с их типом, и каждому из них присваивается значение соответствующего фактического параметра.

Существует два вида передачи величин в функцию:

* по значению:

При передаче по значению в функцию передаются копии значений фактических параметров, и операторы функции работают с этими копиями. Доступа к исходным значениям параметров у функции нет, а, следовательно, нет и возможности их изменить.

* по адресу:

При передаче по адресу в функцию передаются копии адресов параметров, а функция осуществляет доступ к ячейкам памяти по этим адресам и может изменить исходные значения параметров.

При передаче по ссылке в функцию передается адрес указанного при вызове параметра, а внутри функции все обращения к параметру неявно разыменовываются. Поэтому использование ссылок вместо указателей улучшает читаемость программы. Использование ссылок вместо передачи по значению более эффективно, поскольку не требует копирования параметров.

Параметры, передаваемые в функцию, могут быть любого типа, кроме массива или функции, которые передаются с помощью указателей.

Второй способ передачи параметров – это использование величин, объявленных вне всяких функций (глобальные величины). Такие величины видимы для всех функций, описанных после них.

1. Передача функций в качестве параметров функций

Функцию можно вызвать через указатель на нее. Для этого объявляется указатель соответствующего типа и ему с помощью операции взятия адреса присваивается адрес функции:   
<тип> (\*<имя\_указателя>) (<список\_типов\_формальных\_параметров>) = &<имя\_функции>;

void f(int a ){ /\* ... \*/ }//определение функции

void (\*pf)(int);//указатель на функцию

//...

\*pf = &f;// указателю присваивается адрес функции (можно написать pf = f;)

pf(10);// функция f вызывается через указатель pf (можно написать (\*pf)(10) )

typedef

Для того чтобы сделать программу более читаемой, при описании указателей на функции используют переименование типов (typedef). Можно объявлять массивы указателей на функции (это может быть полезно, например, при реализации меню):

Typedef void (\*PF)(int); //описание типа PF как указателя на функцию одним параметром типа int.

Указатели на функции передаются в подпрограмму таким же образом, как и параметры других типов:

void fun(PF pf){ // функция fun получает в качестве параметра указатель типа PF

//…

pf(10); //вызов функции, переданной через указатель

// ...

}

Тип указателя и тип функции, которая вызывается посредством него, должны совпадать в точности.

1. Перегрузка функций в языке С++

Перезагрузкой функции называют создание нескольких функций с одним именем, но разными типами параметров. При этом тип возвращаемого функциями результата не имеет значения. Компилятор определит, какую из функций вызывает пользователь по списку фактических параметров. Такое необходимо, когда один и тот же алгоритм необходимо осуществить для разных типов данных.

* Перегрузка функции позволяет программам определять несколько функций с одним и тем же именем и типом возвращаемого значения.
* Перегрузка функции улучшает удобочитаемость программ
* Перегруженные функции должны возвращать значения одинакового типа, но могут отличаться количеством и типом параметров.
* Перегрузка функций используется обычно для создания нескольких функций с одинаковыми именами, выполняющих сходные задачи, но над разными типами данных.
* Перегруженные функции различаются по их сигнатурам.

Сигнатура – это комбинация имени функции и типов ее параметров (в определенном порядке)

Под перегрузкой функции, понимается определение нескольких функций (две или больше) с одинаковым именем, но различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом. Таким образом, перегрузка функции нужна для того, чтобы избежать дублирования имен функций, выполняющих сходные действия, но с различной программной логикой.

Например, рассмотрим функцию, которая вычисляет площадь прямоугольника:

Float Rectangle (float, float) { //функция, вычисляющая площадь прямоугольника с двумя параметрами

return a\*b;

}

Float Rectangle (float a\_m, float a\_sm, float b\_m, float b\_sm) { // функция, вычисляющая площадь прямоугольника с четырьмя параметрами

return (a\_m\*100+a\_sm) \*(b\_m\*100+b\_sm);

}

Теперь у нас есть две функции, с разной сигнатурой, но одинаковыми именами (перегруженные функции). Сигнатура – это комбинация имени функции и ее параметрами.

Вызов перегруженной функции ничем не отличается от вызова обычных функций (если укажем две переменные сработает первая функция, если четыре, то втора).

1. (стр. 58) Динамические переменные. Выделение и освобождение памяти при выполнении программы.

Динамические переменные – это переменные, которые создаются (и уничтожаются) во время выполнения программы.

В зависимости от того, кто отвечает за работу с такими переменными – компилятор или программист – различают языки с динамическим или статическим управлением памятью.

Динамическое распределение памяти используется для экономии памяти. Те переменные, которые становятся ненужными, уничтожаются, а освобожденное место используется для новых переменных. Это особенно эффективно в задачах, в которых число необходимых объектов зависит от обрабатываемых данных или от действий пользователя, то есть заранее неизвестно. В таких ситуациях можно с запасом предусмотреть место под максимально возможно количество объектов, но крайне ресурсоемко.

Динамическое распределение памяти осуществляется с помощью операций new и delete.

* Операция <тип><имя\_указателя> = new <тип> [(значение)];

возвращает указатель на переменную типа, указанного в угловых скобках, при этом можно сразу же разместить по этому адресу некоторое значение, для этого его достаточно записать в круглых скобках после типа.

* Операция delete<имя\_указателя>;

освобождает область памяти, на которую указывал указатель.

Пример:

Int main(){

Int \*a = new int; // объявление указателя для переменной типа int

Int \*b=new int(5); // инициализация указателя

\*a=10;

\*b=\*a+\*b;

cout << \*b << endl; // выведет 15

delete b;

delete a;

return 0;

}

Для создания и уничтожения динамических переменных в С++ используются операторы new и delete, соответственно.

Значением оператора new является адрес созданной переменной, а в случае неудачи NULL. Созданная переменная не имеет имени и связь с нею можно осуществить только одним способом – присвоив ее адрес указателю соответствующего типа.

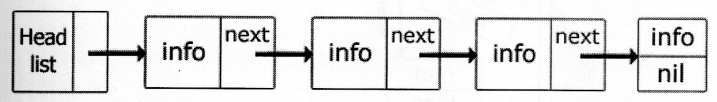
При уничтожении динамической переменной ее адрес (т.е. значение соответствующего указателя) передается оператору delete.

Область видимости динамической переменной совпадает с областью видимости указателя на нее, а время жизни заключено между действием оператора new и оператора delete.

1. (стр. 61) Построение списковых (последовательных) структур

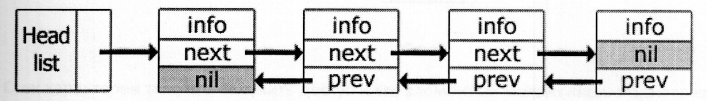
Список – абстрактный тип данных, реализующий упорядоченный набор значений. Списки отличаются от массивов тем, что доступ к их элементам осуществляется последовательно, в то время как массивы – структура данных произвольного доступа. Данный абстрактный тип имеет несколько реализацией в виде структур данных.

Список (связный список) – это структура данных, представляющая собой конечное множество упорядоченных элементов, связанных друг с другом посредствам указателей. Каждый элемент структуры содержит поле с какой-либо информацией, а также указатель на следующий элемент. В отличие от массива, к элементам списка нет произвольного доступа.



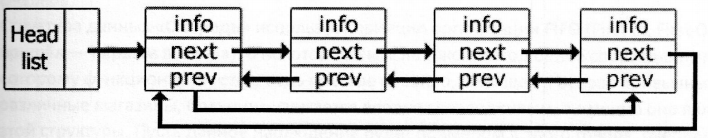
В односвязном списке, приведенным выше, начальный элемент является head list (голова списка), а все остальное называется хвостом. Хвост списка составляют элементы, разделенные на две части: информационную (поле info) и указательную (поле next). В последнем элементе вместо указателя, содержится признак списка – nil.

Односвязный список не слишком удобен, так как из одной точки есть возможность попасть лишь в следующую точку, двигаясь в конец. Когда кроме указателя на следующий элемент есть указатель и на предыдущий, то такой список называется двусвязным.



Возможность двигаться как вперед, так и назад полезна для выполнения некоторых операций, но дополнительные указатели требуют задействования большего количества памяти, чем таковой необходимо в эквивалентном односвязном списке.

Для двух видов списков описанных выше существует подвид, называемый кольцевым списком. Сделать из односвязного списка кольцевой можно добавив всего лишь один указатель в последний элемент, так чтобы он ссылался на первый. А для двусвязного потребуется два указателя: на первый и на последний.



Помимо рассмотренных видов списочных структур есть и другие способы организации данных по типу «список».

Кроме различия по связям, списки делятся по методам работы с данными:

* Стек
* Очередь
* Дек

1. Последовательные, циклические и рекурсивные алгоритмы
2. Базовая структура цифровой машины с хранимой программой
3. Архитектура и структура команд четырёхадресной машины
4. Архитектура и структура команд трёхадресной машины
5. Архитектура и структура команд двухадресной машины
6. Архитектура и структура команд одноадресной машины
7. Классификация устройств памяти систем обработки информации
8. Реализация устройств оперативной и долговременной памяти
9. Позиционные системы счисления и выполнение арифметических операций
10. Представление целых и действительных чисел в позиционных системах счисления
11. Связь между системами счисления с основаниями вида 2K
12. Перевод целых и действительных чисел из одной позиционной системы в другую
13. Прямой, обратный и дополнительный коды
14. Нормализованное представление чисел, переполнение, модифицированные коды
15. Алгоритмы сложения и вычитания в дополнительных и обратных кодах
16. Алгоритмы умножения в дополнительных и обратных кодах
17. Структуры рекурсивных алгоритмов
18. Применение рекурсивных алгоритмов
19. Оценка сложности рекурсивных алгоритмов