**Л Е К Ц И Я 2**

**Основни алгоритмични структури – представянето им със средствата на език за програмиране. Линейни и разклонени алгоритмични структури и представянето им. Особености. Примери.**

В зависимост от алгоритмите, които описват, блок-схемите се разделят на линейни, разклонени и циклични.

1. **Линейни блок-схеми** – при тях липсва блок за анализ и при изпълнението на алгоритмите се преминава задължително през всички блокове. Реализира се чрез алгоритмичната структура - **верига**.

На понятието линеен алгоритъм съответства управляващата линейна структура наречена **“верига”** .

|  |  |  |  |
| --- | --- | --- | --- |
| Начало на алгоритъм | Ръчно въвеждане на данни | Операция | Входно-изходна операция |
| Транзакция (Модификация) | Логически блок | Печат | Препратка |
| Продължение | Конектор | Коментар | Край на алгоритъм |

**Веригата** е вид алгоритмична структура, която включва в себе си строга последователност от действия (оператори), която е една и съща за какъв да е набор от входни данни.

На практика, много рядко се срещат изцяло линейни алгоритми. Веригата обикновено е само малка част от разклонен или цикличен алгоритмичен процес.

Линейна блок-схема (управляваща структура **верига**)

Оператор 1

(действие 1)

Оператор 2

(действие 2)

**...**

Оператор N

(действие N)

1. **Разклонени блок-схеми** – винаги има поне един блок за анализ, който разклонява алгоритъма. В зависимост от стойността на логическия израз – условие се изпълняват различни блокове от блок-схемата. Разклонените блок-схеми се реализират с помощта на структурата за управление **“разклонение”**.

**Разклонението** е структура за управление, която разклонява алгоритмичния процес в две или повече посоки, в зависимост от някакво условие.

Разклонението се нарича **“алтернатива”**, ако се разклонява в две посоки.

**Алтернативата** може да е ***с един изпълним клон*** или ***с два изпълними клона***.

Логически израз

(условие)

Оператор 2

Оператор 1

да

не

Алтернатива с два изпълними клона

Логически израз

(условие)

Оператор 1

да

не

Алтернатива с един изпълним клон

Алгоритъм за присвояване на по-голямата от стойностите **а** и **в** на **z**.

Задачата може да се реши с двете форми на разклонение: пълна (фиг. 1.4а) и непълна (фиг. 1.4.б).

b > z

z :=b

да

z :=a

не

а > в

z :=b

z :=a

да

не

Фиг. 1.4a Фиг. 1.4б

Структура за управление, която разклонява алгоритмичния процес в N+1 посоки , в зависимост от N условия се нарича **«многозначно разклонение»**. Разклонените блок-схеми са съставени от структурите **верига** и **разклонение.**

Селектор К

Оператор 1

Оператор N+1

Оператор 2

Оператор N

не

K=S1

K=S2

K=SN

Многозначно разклонение

1. **Циклични блок-схеми** – налице са група от блокове, които са описани веднъж, но се изпълняват многократно за различни стойности на някои от участващите в тях величини. Цикличните блок-схеми се реализират с помощта на структурата за управление **«цикъл»**.

**Цикълът** е структура за управление, при която част от действията се повтарят многократно за различни стойности на участващите променливи. От цикъла се излиза при изпълнение на определено условие, наречено условие за край. Всеки цикъл трябва да съдържа частите **инициализация** (задаване на начални стойности), **тяло**, **условие за край** и **актуализация** (модификация), независимо от реда, в който са разположени.

* **Инициализация** – задават се начални стойности на величини (променливи), участващи в цикъла. Най-често се среща случая, когато променливата е една, наречена **“управляваща променлива”**.
* **Тяло** – група от блокове, които ще се изпълняват многократно.
* **Актуализация** – променя се стойността на поне една величина от цикъла (най-често това е управляващата променлива).
* **Условие** **за край** – този блок определя дали да се изпълни тялото на цикъла още веднъж или не. Ако условието за край е разположено преди тялото на цикъла, това е **цикъл с предусловие**. Ако условието за край е разположено след тялото на цикъла, това е **цикъл със следусловие**.

***Разликата между цикъл с предусловие и цикъл със следусловие е че този със следусловие се изпълнява задължително поне веднъж, докато този с предусловие може и да не се изпълни.***

Цикличните блок-схеми освен структурата **цикъл** (която е задължителна) могат да включват и структурите **верига** и **разклонение.**

Инициализация

(Задаване на начални стойности)

Тяло на цикъла

Условие

за край

Актуализация

да

не

Блок-схема на цикъл със следусловие

Инициализация

(Задаване на начални стойности)

Тяло на цикъла

Условие

за край

Актуализация

да

не

Блок-схема на цикъл с предусловие

**КЛАСИФИКАЦИЯ НА СОФТУЕРА**

**СИСТЕМЕН СОФТУЕР ПРИЛОЖЕН СОФТУЕР**

ОПЕРАЦИОННИ СИСТЕМИ ПРИЛОЖНИ БИЗНЕС СИСТЕМИ

МРЕЖОВ СОФТУЕР ПРОГРАМНИ ПРОДУКТИ ЗА БИТА

СОФТУЕР ЗА БАЗИ ДАННИ СОФТУЕР ЗА ПРОМИШЛЕНОСТТА

ЕЗИКОВИ ПРОЦЕСОРИ УТИЛИТИ ПРОГРАМИ

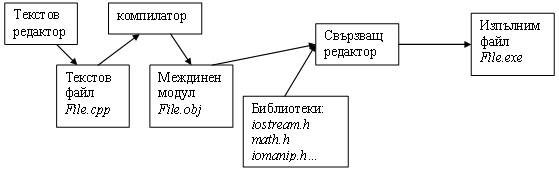
ДРУГИ СИСТЕМНИ СРЕДСТВА ДРУГ ПРИЛОЖЕН СОФТУЕР ------------------------------------------------------------------------------------------------Език за програмиране се нарича съвкупност от семантични конструкции и синтактични правила, наподобяваща човешкия език, която се използва за представяне на структури от данни и алгоритми във форма, удобна за обработка от компютър. Съвкупност от структури и данни и алгоритми, които изпълняват дадена задача се нарича компютърна програма. Хората, които се занимават с писането на програми, се наричат програмисти. Популярни съвременни езици за програмиране са Java, C, C++, PHP, C#, Perl, Python, JavaScript. Формалните алгоритмични езици за програмиране представляват изкуствено създадени езици, които имат следните елементи:

1. азбука(допустимите за езика символи)

2. синтаксис(правила за изграждане на изречения, изрази)

3. семантика(смисъла на изреченията, изразите)

Езиците за програмиране от високо ниво също представляват формални езици за описание на данните и алгоритмите.



Програма- последователност от команди, реализиращи алгоритъм, записани на ЕП. Програмата става изпълнима от компютъра след като бъде преведена на машинен език. Това вършат транслаторите, които биват интерпретатори и компилатори, спред начина по който превеждат. Средите за програмиране включват следните елементи: текстов редактор(за въвеждане текста на програмата), транслатор за превод на програмата в обектен код(obj), свързващ редактор(свързва всички необходими файлове за нашата задача, напр. библиотеките(файлове с готови програмни части)).

**КЛАСИФИКАЦИЯ НА ПРОГРАМНИТЕ ЕЗИЦИ**

**Ниво на езика**

Езиците за програмиране по своите възможности и време на създаване е прието да се делят на няколко поколения (Generation Language, GL). Всяко поколение по своята функционалност качествено се отличава от предидущите.

**Първо поколение** (машинен език) – 40-те години на ХХ век

Машинно-ориентиран език, програмните инструкции, на който използват основния набор инструкции, специфични за всеки процесор. Тези инструкции се наричат машинен код ( обектен код) и се пишат и изпълняват в двоичен код ( 1 и 0). Позволяват на програмиста да кодира инструкциите по възможно най-ефективния начин, но е изключително трудно с тях да се пишат програми.

**Второ поколение** (асемблерен език) -  50-те години на ХХ век

Машинно-ориентиран език от ниско ниво, в който всеки оператор отговаря на инструкция, която процесорът може да изпълни. Вместо двоични кодове, инструкциите използват мнемонични кодове. Обръщението към клетки от паметта е заместено с обръщения към области от паметта. Двоичния код за представяне на числата е заменен с десетичен или шестнадесетичен код. Писането на програми изисква детайлно познаване възможностите на компютъра и неговия микропроцесор.Асемблерите се използват активно и днес за създаване на програми, максимално използващи възможностите на хардуера - различни драйвери, модули за включване на нестандартно оборудване и т.н. В някои области ( например машинната графика) на асемблер се пишат библиотеки, ефективно реализиращи стандартни алгоритми за обработка на данни.

**Трето поколение** (езици от високо ниво) **–** 60-те години на ХХ век

Проблемно и процедурно ориентирани езици, които използват ключови думи и синтаксис, грубо напомнящи човешки език. Те увеличават производителността на програмистите и ги освобождават от необходимостта да знаят по какъв начин се изпълнява всяка инструкция. С тези неща се занимават интерпретатора или компилатора, които преобразуват оригиналния код в изпълним машинен код. Всеки оператор на език от високо ниво отговаря  на няколко инструкции на машинен език. Недостатък – бавна обработка и необходимост от голям обем памет.

**Четвърто поколение -** 90-те години на ХХ век

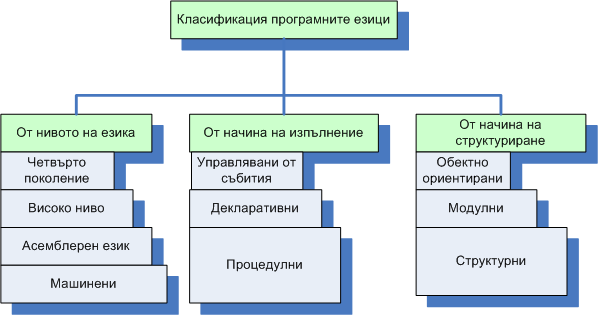
Езиците от четвърто поколение се развиват основно в две направления:

* Сложен език, осигуряващ набор от средства и подпрограми за повишаване производителността на програмистите.
* Достъпен за крайните потребители език, чрез който те могат да създават собствени приложения.

**Характеристики**

* Визуална среда за разработка;
* Обектно-ориентиран подход на програмиране;
* Мощни средства за програмиране на клиент-сървър приложения;
* Средства за управление на бази данни.

Притежават развит потребителски интерфейс, позволяващ диалогова работа при разработка на приложения, при определяне на входните и изходни екрани и отчети, при определяне съдържанието на записите и при употреба на графики.



**Процедурни (алгоритмични езици) езици**

Програмистът описва различните стъпки от процедурата, която компютърът трябва да следва, за да изпълни задачата.

*Примери-Асемблер,Фортран,Паскал,Си.*

**Декларативни езици**

Дефинират се множество от факти и взаимоотношения, като се очакват определени резултати. Използването им е ограничено до експертни системи и справочни приложения за бази данни.

*Примери - Пролог, Лисп и др.*

**Езици, управлявани от събития**

Програмен език, чрез който се създават програми чрез предизвикване на събитие. Такива програми пускат компютъра в безкраен цикъл, докато не се случи някакво събитие, по време на  което се обработва кода управляващ събитието. По този начин програмата директно реагира на събития предизвикани от потребителя, без да преминава през серия от диалози и менюта по строго определен начин.

*Пример: Visual Basic*

**Начин на структуриране**

**Структурно програмиране**  - набор от стандарти, които правят програмите по дълги, но улесняват четливостта, надеждността и поддръжката им. Поддържа се идеята,че цялата структура на програмата трябва да отразява логическата последователност на извършваните действия, започвайки с първата задача и продължавайки логически. За езици като  Си, Паскал, Модула 2 и dBase е присъщо  структурирането.

**Модулно програмиране**

Стил на програмиране ,който разбива програмните функции на модули, всеки от които изпълнява една функция и съдържа всички кодове и променливи за изпълнението на тази функция.

**Обектно ориентиран език за програмиране**

Непроцедурен език, в който програмните елементи са във вид на обекти, които могат да си предават съобщения един на друг. В една обектно ориентирана програма, всеки обект има свои собствени данни от програмен код и вътрешно разчита на себе си. Програмата прави обекта част от голямото цяло чрез обединяването му в йерархия от слоеве. Обектно ориентираното програмиране е първично разширение на концепцията за модулно програмиране.

**История на езика С**

Създаден през 1972 в „AT&T Bell Laboratories“ от Dennis Ritchie с цел създаване на операционната система Unix. Разработката на езика е тясно свързана с операционната система UNIX, чието ядро впоследствие бива пренаписано на езика C. C е подходящ за създаване както на операционни системи, така и на приложения. Много от днешните операционни системи ОС са написани главно на C. Популярните езици C++ и Objective-C са разширения на езика С. Код, написан на С, може много лесно да се използва на различни платформи. За езика са разработени няколко стандарта - KNR, ANSI C и C99. Някои от базовите характеристиките на езика са:

• Слабо типизиран (weak typing) - Позволява автоматично преобразуване между някои типове данни.

• Статично типизиран (static typing) - Типовете на данните се определят по време на компилирането.

• Позволява работа с паметта на ниско ниво - поддържа указатели към променливи, указатели към функции и адресна аритметика.

• Използва сравнително малък набор от ключови думи, като за сметка на това използва сравнително голямо количество оператори.

**Допълнителни характеристики на езика C**

Мощен език с възможности за достъп до ниско ниволесно включване на програми на асемблер, манипулация на битове, директен достъп до адреси от паметта чрез указатели, адресна аритметика

Преносимост базирана на прекомпилация на програмата.

Възможност за създаване и манипулиране на най-разнообразни потребителски типове данни, широк набор от готови функции.

Ограничен контрол върху типовете данни.

Създаване на изключително ефективни програми от гледна точка на памет и бързодействие.Лесно създаване и комбиниране на програмни модули, които позволяват отделна настройка и комбинирането им за създаване на мощни програми.

**Фази на компилатора за език С**

1. Лексически или лексикален анализ

2. Синтактичен анализ

3. Семантичен анализ

4. Генериране на междинен код

5. Оптимизация

6. Генериране на код

**ПРОГРАМА НА С**

Програма на С: данни + алгоритъм за обработката им

Структура на програма на С – набор от равнопоставени програмни единици функции. С една от тях (int main() ) започва изпълнението на програмата (задава входна точка ). Всяка функция представлява независима програмна единица с изчисляван резултат, име и параметри (входни данни за функцията). Всяка една функция може да извиква всяка друга функция в програмата, като след завършването управлението се връща в извикващата функция. Функциите не могат да се включват една в друга.

СТРУКТУРА НА ПРОГРАМА НА С

\* Команди на препроцесора

\* Дефиниране на типове

\* Прототипи на функции–декларира типа на функцията и параметрите й

\* Променливи

\* Функции->

\* заглавен ред (тип, име, формални параметри)

\* променливи

\* изпълними инструкции

**(0) #include <stdio.h>**

**(1) typedef**

**(2) int chng1(int, int);**

**(3) int chng2(int,int);**

**(4) int chng1(int p, int delta){**

**(5) return p+delta; /\*addition \*/**

**(6) }**

**(7) int main() {**

**(8) int a =5,b;**

**(9) a = chng1(a,3); // call first function**

**(10) b = chng2(a,1);**

**(11) return a;**

**(12) }**

**(13) int chng2(int k,int delta){**

**(14) return k-delta; /\*subtraction\*/**

**(15) }**

**Ред на изпълнение:** (8) -> (9)->(5)->(9)->(10)->(14) ->(10)->(11)->край на програмата. Езикът прави разлика между главни малки букви (case sensitive) –W и w са 2 различни букви.

**ПРЕДСТАВЯНЕ НА ДАННИ В ЕЗИКА С**

Характеристики на данните:

Тип - възможни стойности, начин на представяне в паметта, възможни операции

клас памет – цикъл на живот (възникване, начална стойност, разрушаване),видимост

**Константи и Променливи**

Константите представляват данни, които фиксират определени стойности в началото на програмата и запазват тези стойности по време на изпълнението на програмата до завършването й. Константите могат да бъдат задавани със стойността си(литерали) или да бъдат дефинирани с тип и име.

Променливите - както показва и наименованието им- представляват данни, които могат да бъдат променяни по време на изпълнение на програмата. Името на променливата в езика С може да включва от един до 8 символа, като имената с повече символа се обработват от компилатора до осмия символ и следващите се игнорират.Всяка променлива преди да бъде използвана, трябва да бъде дефинирана с типа си.

Пример за константа: **const double pi = 3.1415;**

Пример за променливa: **long int counter;**

**Основни типове данни в езика С**

Типовете данни зависят от конкретния компилатор който се използва. Данните показани по-долу са примерни за 32bit Intel съвместими процесори.

**Тип Вид данни Интервал**

|  |  |  |
| --- | --- | --- |
| char | Основната адресируема единица в машината — байт. В C се използва предимно за работа със символи. | от –128 до 127 |
| unsigned char | Основната адресируема единица в машината — байт. В C се използва предимно за работа със символи. | от 0 до 255 |
| short | Цели числа (ℤ) | от –32 768 до 32 767 |
| unsigned short | Естествени числа ( ℕ) | 0 до 65 535 |
| int, long | Цели числа ( ℤ) | –2 147 483 648 до 2 147 483 647 |
| unsigned int, unsigned long | Естествени числа ( ℕ) | от 0 до 4 294 967 295 |
| float | Число с плаваща запетая. | от ± 1,40239846×10−45 до ±3,40282347×1038 |
| double | Число с плаваща запетая с двойна прецизност. | от ±4,94065645841246544×10−324 до ±1,79769313486231570×10308 |
| void | неопределен тип | — |

ПРИМЕРИ:

**%d print as decimal integer**

**%6d print as decimal integer, at least 6 characters wide**

**%f print as floating point**

**%6f print as floating point, at least 6 characters wide**

**%.2f print as floating point, 2 characters after decimal point**

**%6.2f print as floating point, at least 6 wide and 2 after decimal points**

Стандартът ANSI C определя 32 ключови думи, които не могат да се използват в имената на функции или променливи. Много компилатори на С добавят други ключови думи. В С главните и малките букви се различават (int, Int и INT са различни).

Запазени думи:

**auto, bool, break, case, char, complex, const, continue, default, do, double, else, enum, extern, float, for, goto, if, Imaginary, inline, int, long, register, restrict, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while**

В следващия пример, на екрана се извежда текст Hello, World! („Здравей, Свят!“), целящ да покаже синтаксиса на С:

**#include <stdio.h>**

**int main(void)**

**{**

**printf("Hello, World!\n");**

**return 0;**

**}**

**C99**

C99 е нов стандарт на езика, утвърден през 1999 от ISO/IEC, добавящ някои важни и възможности, между които са:

* Inline функции
* Премахване ограничението за деклариране на променливи само в началото на блок
* Нови типове данни, включващи long long int, boolean и др.
* Променлива дължина на масивите
* Поддръжка на едноредови коментари //, като в някои други езици за програмиране
* Нови библиотечни функции
* Нови заглавни файлове, като stdbool.h и inttypes.h
* Подобрена поддръжка на IEEE floating point
* Добавяне на нови ключови думи, като restrict

**Максимални и минимални стойности реални:**

**#include <stdio.h>**

**#include <float.h>**

**Int main() {**

**printf("Minimum value of a float: %g\n", FLT\_MIN );**

**printf("Maximum value of a float: %g\n", FLT\_MAX );**

**printf("Precision of a float: %d digits\n", FLT\_DIG );**

**printf("Minimum value of a double: %g\n", DBL\_MIN );**

**printf("Maximum value of a double: %g\n", DBL\_MAX );**

**printf("Precision of a double: %d digits\n", DBL\_DIG );**

**}**

**Minimum value of a float: 1.17549e-038**

**Maximum value of a float: 3.40282e+038**

**Precision of a float: 6 digits**

**Minimum value of a double: 2.22507e-308**

**Maximum value of a double: 1.79769e+308**

**Precision of a double: 15 digits.**

**ПРИМЕРИ**

**int limit = MAXLINE+1;**

**const double e = 2.71828182845905;**

**float eps = 1.0e-5;**

**char m;**

**int high,low;**

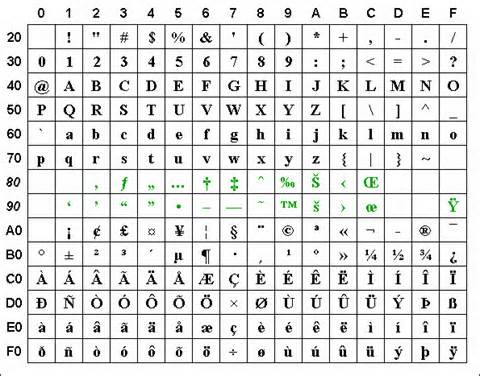
**int high=17,low=10;**

**char c='b'; ( '/0142' 98**10**)**

**short int sh;**

**long int counter;**

В последните два примера **int** може да се пропусне.



По-горе е показана ASCII таблица на допустимите символи и техните десетични еквиваленти. Например, символът **b** е с десетичен код 62. Азбуката, която е допустима в езика С е :

**Главни и малки латински символи: a–z A–Z**

**Десетични символи: 0–9**

**Графични символи: ! " # % & ' ( ) \* + , - . / : ; < = > ? [ \ ] ^ \_ { | } ~**

Всички възможни базови аритметични и символни типове са показани в таблицата по-долу:

|  |  |  |
| --- | --- | --- |
| **ТИП** | **СЪДЪРЖАНИЕ СПЕСИФИКАТОР** |  |
| **char** | Smallest addressable unit of the machine that can contain basic character set. It is an integer type. Actual type can be either signed or unsigned depending on the implementation. | **%c** |
| **signed char** | Of the same size as char, but guaranteed to be signed. | **%c (or %hhi for numerical output)** |
| **unsigned char** | Of the same size as char, but guaranteed to be unsigned. | **%c (or %hhu for numerical output)** |
| **short short int signed short signed short int** | *Short* signed integer type. Capable of containing at least the [−32767,+32767] range;[[3]](https://en.wikipedia.org/wiki/C_data_types#cite_note-c99sizes-3) thus, it is at least 16 bits in size. | **%hi** |
| **unsigned short unsigned short int** | The same as short, but unsigned. | **%hu** |
| **int signed int** | Basic signed integer type. Capable of containing at least the [−32767,+32767] range;[[3]](https://en.wikipedia.org/wiki/C_data_types#cite_note-c99sizes-3) thus, it is at least 16 bits in size. | **%i or %d** |
| **unsigned unsigned int** | The same as int, but unsigned. | **%u** |
| **long long int signed long signed long int** | *Long* signed integer type. Capable of containing at least the [−2147483647,+2147483647] range;[[3]](https://en.wikipedia.org/wiki/C_data_types#cite_note-c99sizes-3) thus, it is at least 32 bits in size. | **%li** |
| **unsignedlong unsigned long int** | The same as long, but unsigned. | **%lu** |
| **long long long long int signed long long signed long long int** | *Long long* signed integer type. Capable of containing at least the [−9223372036854775807,+9223372036854775807] range;[[3]](https://en.wikipedia.org/wiki/C_data_types#cite_note-c99sizes-3) thus, it is at least 64 bits in size. Specified since the [C99](https://en.wikipedia.org/wiki/C99) version of the standard. | **%lli** |
| **unsigned long long unsigned long long int** | The same as long long, but unsigned. Specified since the [C99](https://en.wikipedia.org/wiki/C99) version of the standard. | **%llu** |
| **float** | Real floating-point type, usually referred to as a single-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the [IEEE 754 single-precision binary floating-point format](https://en.wikipedia.org/wiki/Single-precision_floating-point_format). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic". | **%f (promoted automatically to double for printf())** |
| **double** | Real floating-point type, usually referred to as a double-precision floating-point type. Actual properties unspecified (except minimum limits), however on most systems this is the [IEEE 754 double-precision binary floating-point format](https://en.wikipedia.org/wiki/Double-precision_floating-point_format). This format is required by the optional Annex F "IEC 60559 floating-point arithmetic". | **%f (%lf for scanf())** |
| **long double** | Real floating-point type, usually mapped to an [extended precision](https://en.wikipedia.org/wiki/Extended_precision) floating-point number format. Actual properties unspecified. Unlike types float and double, it can be either [80-bit floating point format](https://en.wikipedia.org/wiki/80-bit_floating_point_format), the non-IEEE "[double-double](https://en.wikipedia.org/wiki/Double-double_arithmetic)" or [IEEE 754 quadruple-precision floating-point format](https://en.wikipedia.org/wiki/IEEE_754_quadruple-precision_floating-point_format) if a higher precision format is provided, otherwise it is the same as double. | **%Lf** |