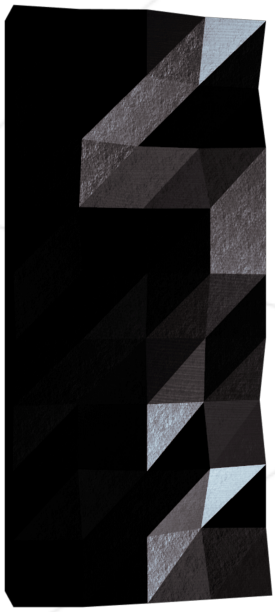

Libft

Tu primera librería

Resumen: *Este proyecto consiste en crear una librería en* `C` *que contendrá una serie de funciones de propósito general que tus programas podrán utilizar.*

Versión: 19.0

Índice general

Capítulo I

Introducción

Programar en `C` puede ser tedioso cuando no se tiene acceso a las funciones estándar más utilizadas. Este proyecto tiene como objetivo ayudarte a comprender cómo actuán estas funciones, implementarlas por tu cuenta y aprender a utilizarlas de forma eficaz. Desarrollarás una librería propia que te será muy útil en los próximos proyectos de `C`.

Asegúrate de ir enriqueciendo tu `libft` a lo largo del cursus. Sin embargo, cuando la utilices, asegúrate de que todas las funciones empleadas por tu librería respetan las permitidas por cada proyecto.

Capítulo II

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) excepto en el caso de comportamientos indefinidos. Si esto sucede, tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, utilizar cc y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo III

Instrucciones sobre la IA

Contexto

Este proyecto está diseñado para ayudarte a descubrir los fundamentos que construirán tu formación en TIC (lo que conocemos como Tecnologías de la Información y la Comunicación).

Para afianzar los conocimientos y habilidades clave, es esencial adoptar un enfoque reflexivo sobre el uso de herramientas de IA.

El auténtico aprendizaje de los fundamentos requiere un esfuerzo intelectual real, a través de desafíos, repetición y el intercambio de conocimiento que procede del aprendizaje entre pares.

Para una visión más completa de nuestra postura sobre la IA (ya sea como herramienta de aprendizaje, como parte del plan de estudios de TIC o como una expectativa en el mercado laboral) puedes consultar las preguntas frecuentes dedicadas en la intranet.

Mensaje principal:

- Construir fundamentos sólidos sin atajos.
- Desarollar de forma real habilidades técnicas y transversales.
- Experimentar el aprendizaje entre pares de forma inmersiva, empezando por aprender a aprender y por resolver nuevos problemas.
- El proceso de aprendizaje es más importante que el resultado.
- Aprender sobre los riesgos asociados a la IA y desarrollar prácticas de control efectivas y medidas que neutralicen los errores comunes.

Reglas para estudiantes:

- Aplica la lógica y el razonamiento a las tareas asignadas, especialmente antes de recurrir a la IA.
- No deberías pedir respuestas directas a la IA.
- Infórmate sobre el enfoque global de 42 respecto la IA.

Resultados de esta etapa:

Durante esta fase de construcción de los fundamentos, conseguirás:

- Obtener una base adecuada en tecnología y en programación.
- Comprender por qué y cómo la IA puede ser peligrosa durante esta fase.

Comentarios y ejemplos:

- Sí, sabemos que la IA existe.Y si, también sabemos que puede resolver tus proyectos. Pero estás aquí para aprender, no para demostrar que la IA ha aprendido. No pierdas tiempo solo para demostrar que la IA puede resolver un problema determinado.
- Aprender en 42 no tiene nada que ver con saber una respuesta, sino con desarrollar las habilidades para encontrarla. La IA te dará la respuesta directa, lo que impide que desarrolles tu propio razonamiento. Razonar requiere tiempo, esfuerzo y cometer errores. Nadie dijo que el proceso iba a sér fácil.
- Ten en cuenta que, durante los exámenes, no tendrás acceso a la IA (no tenemos internet ni dispositivos inteligentes). Te vas a dar cuenta rápidamente si has confiado demasiado en la IA durante tu proceso de aprendizaje de la forma más directa: frente a una hoja en blanco donde vas a tener que escribir tu propio código.
- El aprendizaje entre pares te expone a diferentes ideas y enfoques, mejorando tus habilidades transversales y tu capacidad de pensar de forma diferente. Eso es mucho más valioso que sentarte a chatear con un bot. Así que, ¡sin miedo! Habla, haz preguntas y aprende con el resto de estudiantes.
- Sí, la IA formará parte del plan de estudios, tanto como herramienta de aprendizaje como tema en sí mismo. Incluso tendrás la oportunidad de crear tu propio software de IA. Para aprender más sobre nuestro enfoque progresivo, puedes consultar la documentación disponible en la intranet.

✓ Buenas prácticas:

Me atasco en un nuevo concepto. Le pregunto a alguien cercano cómo lo ha abordado. Hablamos durante 10 minutos y, de repente, todo encaja. Lo entiendo. No entiendo algo concreto del proyecto y no sé cómo continuar. Le pregunto a otra persona cómo lo ha abordado, hablamos sobre el tema y, si es necesario, incluso utilizamos otros métodos (papel y boli, dibujos, metáforas, etc.) hasta conseguir entenderlo.

✗ Mala práctica:

Utilizo la IA en secreto, copio un código que parece correcto. Durante la evaluación entre pares, no puedo explicar nada. Suspendo. Durante el examen, sin IA, me vuelvo a atascar. Suspendo.

Capítulo IV

Parte obligatoria

Nombre de programa	`libft.a`
Archivos a entregar	`Makefile, libft.h, ft_*.c`
Makefile	`NAME, all, clean, fclean, re`
Funciones autorizadas	`Detalles debajo`
Se permite usar libft	`Todavía no disponible`
Descripción	`Escribe tu propia librería: un conjunto de funciones que será una herramienta muy útil a lo largo del cursus.`

IV.1. Consideraciones técnicas

- `Declarar` variables globales está prohibido.
- Si necesitas crear funciones auxiliares para descomponer una función más compleja, debes definirlas como `static` para restringir su alcance al archivo correspondiente.
- Todos los archivos deben colocarse en la raíz de tu repositorio.
- Se prohíbe entregar archivos no utilizados.
- Todos los archivos `.c` deben compilarse con las opciones (*flags*) `-Wall -Werror -Wextra`.
- Debes utilizar el comando `ar` para generar la librería. El uso de `libtool` queda prohibido.
- Tu `libft.a` debe ser creado en la raíz de tu repositorio.

IV.2. Parte 1 - Funciones de libc

Para empezar, deberás rehacer algunas funciones de la `libc`. Tus funciones tendrán los mismos prototipos y comportamientos que las funciones originales, siguiendo fielmente las definiciones indicadas en la página del `man` de cada función. La única diferencia será el nombre: todas deberán comenzar con el prefijo `"ft_"`. Por ejemplo, `strlen` se convertirá en `ft_strlen`.

Algunas funciones tienen en sus prototipos la palabra "restrict". Esta palabra forma parte del estándar de c99. Por lo tanto, está prohibido incluirla en tus propios prototipos, así como compilar tu código con la opción -std=c99.

Deberás escribir tus propias versiones de las siguientes funciones originales. No deben depender de ninguna función externa.

Para las funciones de clasificación de caracteres (isalpha, isdigit, isalnum, isascii, isprint), el valor de retorno debe ser:

- 1 si el carácter cumple la condición comprobada.
- 0 si el carácter no la cumple.

- isalpha
- isdigit
- isalnum
- isascii
- isprint
- strlen
- memset
- bzero
- memcpy
- memmove
- strlcpy
- strlcat
- toupper
- tolower
- strchr
- strrchr
- strncmp
- memchr
- memcmp
- strnstr
- atoi

Para implementar estas otras dos funciones, tendrás que utilizar `malloc()`:

- calloc
- strdup

Dependiendo de tu sistema operativo actual, el comportamiento de la función calloc puede diferir de lo descrito en su página del manual. Debes seguir la siguiente regla: Si nmemb o size es 0, entonces calloc() debe devolver un puntero único que pueda pasarse con éxito a free().

Algunas de las funciones que debes rehacer, como strlcpy, strlcat y bzero, no están incluidas por defecto en la biblioteca GNU C (glibc). Para compararlas con la versión estándar del sistema, puede ser necesario incluir <bsd/string.h> y compilar con la opción -lbsd. Este comportamiento es propio de los sistemas basados en glibc. Si tienes interés, vale la pena aprovechar la ocasión para explorar las diferencias entre glibc y la biblioteca libc de BSD.

IV.3. Parte 2 - Funciones adicionales

En esta segunda parte, deberás desarrollar un conjunto de funciones que, o no son de la librería `libc`, o lo son pero de una forma distinta.

Algunas de las siguientes funciones pueden ser útiles para hacer las funciones de la parte 1.

Nombre de función	ft_substr
Prototipo	char *ft_substr(char const *s, unsigned int start, size_t len);
Archivos a entregar	-
Parámetros	s: La cadena original desde la que se crea la subcadena. start: El índice del carácter en 's' desde el que empieza la subcadena. len: La longitud máxima de la subcadena.
Valor devuelto	La subcadena de caracteres resultante. *NULL* si falla la reserva de memoria.
Funciones autorizadas	malloc
Descripción	Reserva memoria (con malloc(3)) y devuelve una subcadena de caracteres de la cadena 's'. La subcadena comienza en el índice '*start*' y tiene una longitud máxima '*len*'.

Nombre de función	ft_strjoin
Prototipo	char *ft_strjoin(char const *s1, char const *s2);
Archivos a entregar	-
Parámetros	s1: La primera cadena. s2: La cadena a añadir a 's1'.
Valor devuelto	La nueva cadena de caracteres. *NULL* si falla la reserva de memoria.
Funciones autorizadas	malloc
Descripción	Reserva memoria (con malloc(3)) y devuelve una nueva cadena de caracteres, formada por la concatenación de 's1' y 's2'.

Nombre de función	`ft_strtrim`
Prototipo	`char *ft_strtrim(char const *s1, char const *set);`
Archivos a entregar	-
Parámetros	`s1: La cadena de caracteres que debe ser recortada.` `set: Los caracteres a eliminar de la cadena en cuestión.`
Valor devuelto	`La cadena recortada resultante.` *NULL* `si falla la reserva de memoria.`
Funciones autorizadas	`malloc`
Descripción	`Reserva memoria (con malloc(3)) y devuelve una copia de 's1' con los caracteres de '`*set*`' eliminados al principio y al final.`

Nombre de función	`ft_split`
Prototipo	`char **ft_split(char const *s, char c);`
Archivos a entregar	-
Parámetros	`s: La cadena que se va a dividir.` `c: El carácter delimitador.`
Valor devuelto	`El arreglo de nuevas cadenas resultante de la separación.` *NULL* `si falla la reserva de memoria.`
Funciones autorizadas	`malloc, free`
Descripción	`Reserva memoria (utilizando malloc(3)) y devuelve un arreglo (`*array*`) de cadenas obtenido al dividir la cadena 's' en subcadenas utilizando el carácter 'c' como delimitador. El arreglo debe terminar con un puntero a` *NULL*`.`

Nombre de función	`ft_itoa`
Prototipo	`char *ft_itoa(int n);`
Archivos a entregar	-
Parámetros	`n: el entero a convertir.`
Valor devuelto	`La cadena que represente el número.` *NULL* `si falla la reserva de memoria.`
Funciones autorizadas	`malloc`
Descripción	`Reserva memoria (utilizando malloc(3)) y devuelve una cadena que represente el valor del número entero recibido como argumento. Debe ser capaz de manejar números negativos.`

Nombre de función	`ft_strmapi`
Prototipo	`char *ft_strmapi(char const *s, char (*f)(unsigned int, char));`
Archivos a entregar	-
Parámetros	`s: La cadena sobre la que iterar.` `f: La función a aplicar sobre cada carácter.`
Valor devuelto	`La cadena creada tras el correcto uso de 'f' sobre cada carácter.` *NULL* `si falla la reserva de memoria.`
Funciones autorizadas	`malloc`
Descripción	`Aplica la función 'f' a cada carácter de la cadena 's', pasando su índice como primer argumento y el propio carácter como segundo argumento. Se crea una nueva cadena (utilizando malloc(3)) para almacenar los resultados de las sucesivas aplicaciones de 'f'.`

Nombre de función	`ft_striteri`
Prototipo	`void ft_striteri(char *s, void (*f)(unsigned int, char*));`
Archivos a entregar	-
Parámetros	`s: La cadena sobre la que iterar.` `f: La función a aplicar sobre cada carácter.`
Valor devuelto	`Nada`
Funciones autorizadas	`Ninguna`
Descripción	`Aplica la función 'f' a cada carácter de la string 's', pasando como parámetros el índice de cada carácter dentro de 's' y la dirección del propio carácter, que puede modificarse si es necesario.`

Nombre de función	`ft_putchar_fd`
Prototipo	`void ft_putchar_fd(char c, int fd);`
Archivos a entregar	-
Parámetros	`c: El carácter a enviar.` `fd: El descriptor de archivo sobre el que escribir.`
Valor devuelto	`Nada`
Funciones autorizadas	`write`
Descripción	`Envía el carácter 'c' al descriptor de archivo (`*`file descriptor`*`) especificado.`

Nombre de función	`ft_putstr_fd`
Prototipo	`void ft_putstr_fd(char *s, int fd);`
Archivos a entregar	-
Parámetros	`s: La cadena a enviar.` `fd: El descriptor de archivo sobre el que escribir.`
Valor devuelto	`Nada`
Funciones autorizadas	`write`
Descripción	`Envía la cadena 's' al descriptor de archivo especificado.`

Nombre de función	`ft_putendl_fd`
Prototipo	`void ft_putendl_fd(char *s, int fd);`
Archivos a entregar	-
Parámetros	`s: La cadena a enviar.` `fd: El descriptor de archivo sobre el que escribir.`
Valor devuelto	`Nada`
Funciones autorizadas	`write`
Descripción	`Envía la cadena 's' al descriptor de archivo dado, seguido de un salto de línea.`

Nombre de función	`ft_putnbr_fd`
Prototipo	`void ft_putnbr_fd(int n, int fd);`
Archivos a entregar	-
Parámetros	`n: El número entero que se envía.` `fd: El descriptor de archivo sobre el que escribir.`
Valor devuelto	`Nada`
Funciones autorizadas	`write`
Descripción	`Escribe el número entero 'n' en el descriptor de archivo dado.`

IV.4. Part 3 - Listas enlazadas

Las funciones para manejar memoria y cadenas son muy útiles, pero pronto descubrirás que trabajar con listas puede serlo aún más.

En esta tercera parte deberás implementar funciones que utilicen una estructura para manejar listas enlazadas. Para ello, añade la siguiente declaración de estructura a tu archivo `libft.h`:

```
typedef struct      s_list
{
  void              *content;
  struct s_list     *next;
}                   t_list;
```

Las variables de la estructura `t_list` son:

- `content`: los datos contenidos en el nodo.
  Usar `void *` permite almacenar cualquier tipo de dato.
- `next`: la dirección del siguiente nodo, o *NULL* si el nodo actual es el último de la lista.

Implementa las siguientes funciones para poder utilizar tus listas de manera sencilla:

Nombre de función	`ft_lstnew`
Prototipo	`t_list *ft_lstnew(void *content);`
Archivos a entregar	-
Parámetros	`content: el contenido con el que se crea el nodo.`
Valor devuelto	`Un puntero al nuevo nodo`
Funciones autorizadas	`malloc`
Descripción	`Reserva memoria (usando malloc(3)) y devuelve un nuevo nodo. La variable 'content' se inicializa con el contenido del parámetro 'content'. Mientras que la variable 'next' se inicializa con` *NULL*.

Nombre de función	ft_lstadd_front
Prototipo	void ft_lstadd_front(t_list **lst, t_list *new);
Archivos a entregar	-
Parámetros	lst: la dirección de memoria de un puntero al primer nodo de una lista. new: un puntero al nodo que se añade al principio de la lista.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Añade el nodo 'new' al principio de la lista 'lst'.

Nombre de función	ft_lstsize
Prototipo	int ft_lstsize(t_list *lst);
Archivos a entregar	-
Parámetros	lst: el principio de la lista.
Valor devuelto	La longitud de la lista.
Funciones autorizadas	Ninguna
Descripción	Cuenta el número de nodos de una lista.

Nombre de función	ft_lstlast
Prototipo	t_list *ft_lstlast(t_list *lst);
Archivos a entregar	-
Parámetros	lst: el principio de la lista.
Valor devuelto	Último nodo de la lista.
Funciones autorizadas	Ninguna
Descripción	Devuelve el último nodo de la lista.

Nombre de función	ft_lstadd_back
Prototipo	void ft_lstadd_back(t_list **lst, t_list *new);
Archivos a entregar	-
Parámetros	lst: el puntero al primer nodo de una lista. new: el puntero a un nodo que se añade a la lista.
Valor devuelto	Nada
Funciones autorizadas	Ninguna
Descripción	Añade el nodo 'new' al final de la lista 'lst'.

Nombre de función	ft_lstdelone
Prototipo	void ft_lstdelone(t_list *lst, void (*del)(void *));
Archivos a entregar	-
Parámetros	lst: el nodo a liberar. del: un puntero a la función utilizada para liberar el contenido del nodo.
Valor devuelto	Nada
Funciones autorizadas	free
Descripción	Recibe como parámetro un nodo 'lst' y libera la memoria del contenido utilizando la función 'del' dada como parámetro. También libera el nodo en sí mismo, pero no libera el siguiente nodo.

Nombre de función	ft_lstclear
Prototipo	void ft_lstclear(t_list **lst, void (*del)(void *));
Archivos a entregar	-
Parámetros	lst: la dirección de un puntero a un nodo. del: un puntero a la función utilizado para eliminar el contenido de un nodo.
Valor devuelto	Nada
Funciones autorizadas	free
Descripción	Elimina y libera el nodo 'lst' dado y todos los consecutivos del mismo, utilizando la función 'del' y free(3). Al final, el puntero a la lista debe ser *NULL*.

Nombre de función	`ft_lstiter`
Prototipo	`void ft_lstiter(t_list *lst, void (*f)(void *));`
Archivos a entregar	-
Parámetros	`lst: un puntero al primer nodo.` `f: un puntero a la función que utilizará cada nodo.`
Valor devuelto	`Nada`
Funciones autorizadas	`Ninguna`
Descripción	`Itera la lista 'lst' y aplica la función 'f' en el contenido de cada nodo.`

Nombre de función	`ft_lstmap`
Prototipo	`t_list *ft_lstmap(t_list *lst, void *(*f)(void *), void (*del)(void *));`
Archivos a entregar	-

Parámetros	`lst: un puntero a un nodo.`
	`f: la dirección de un puntero a una función usada en la iteración de cada elemento de la lista.`
	`del: un puntero a función utilizado para eliminar el contenido de un nodo, si es necesario.`

Valor devuelto	`La nueva lista.` *`NULL`* `si falla la reserva de memoria.`
Funciones autorizadas	`malloc, free`
Descripción	`Itera la lista 'lst' y aplica la función 'f' al contenido de cada nodo. Crea una lista resultante de aplicar sucesivamente la función 'f' a cada nodo. La función 'del' se utiliza para eliminar el contenido de un nodo si es necesario.`

Capítulo V

Requisitos del Readme

Debe incluirse un archivo `README.md` en la raíz del repositorio Git. Su propósito es permitir que cualquier persona que no esté familiarizada con el proyecto (pares, personal, responsables de selección, etc.) pueda entender rápidamente de qué trata el proyecto, cómo ejecutarlo y dónde encontrar más información sobre el tema.
El `README.md` debe incluir, como mínimo:

- La primera línea debe estar en cursiva y decir: *Este proyecto ha sido creado como parte del currículo de 42 por <login1>[, <login2>[, <login3>[...]]].*
- Una sección de "**Descripción**"que presente claramente el proyecto, incluyendo su objetivo y una breve visión general.
- Una sección de "**Instrucciones**"que contenga cualquier información relevante sobre compilación, instalación y/o ejecución.
- Una sección de "**Recursos**"que enumere referencias clásicas relacionadas con el tema (documentación, artículos, tutoriales, etc.), así como una descripción del uso de IA, especificando para qué tareas y en qué partes del proyecto se ha utilizado.

➠ **Podrían requerirse secciones adicionales dependiendo del proyecto** (por ejemplo, ejemplos de uso, lista de características, decisiones técnicas, etc.).

Cualquier contenido extra requerida se listará explícitamente a continuación.

- También debe incluirse una descripción detallada de la librería creada para este proyecto.

`La elección del idioma queda a tu criterio. Se recomienda escribir en inglés, pero no es obligatorio.`

Capítulo VI

Entrega y evaluación

Entrega tu proyecto en tu repositorio `Git` como de costumbre. Solo el trabajo entregado dentro del repositorio será evaluado durante la defensa. No dudes en comprobar varias veces los nombres de los archivos para verificar que sean correctos.

`Deja todos tus archivos en la raíz del repositorio.`

Durante la evaluación, es posible que se solicite una breve **modificación del proyecto**. Esto puede consistir en ajustar ligeramente el comportamiento, modificar unas cuantas líneas de código o incorporar una característica fácil de implementar.

Puede que este paso **no sea necesario en todos los proyectos**, pero debes tenerlo en cuenta si asi se especifica en la hoja de evaluación.

Este paso sirve para a verificar tu comprensión real de una parte específica del proyecto. La modificación se puede realizar en cualquier entorno de desarrollo que elijas (por ejemplo, tu configuración habitual), y debería ser factible en unos pocos minutos, a menos que se defina un plazo específico como parte de la evaluación.
Por ejemplo, se te puede pedir hacer una pequeña actualización en una función o script, modificar lo que se vería en pantalla o ajustar una estructura de datos para almacenar nueva información, etc.

Los detalles (alcance, objetivo, etc.) se especificarán cada **hoja de evaluación** y pueden

variar de una evaluación a otra para el mismo proyecto.

Rnpu cebwrpg va gur 42 Pbzzba Pber pbagnvaf na rapbqrq uvag. Sbe rnpu pvepyr, bayl bar cebwrpg cebivqrf gur pbeerpg uvag arrqrq sbe gur arkg pvepyr. Guvf punyyratr vf vaqvivqhny, jvgu n svany cevmr sbe bar fghqrag. Fgnss zrzoref znl cnegvpvcngr ohg ner abg ryvtvoyr sbe n cevmr. Ner lbh nzbat gur irel svefg gb fbyir n pvepyr? Fraq gur uvagf jvgu rkcynangvbaf gb by@42.se gb or nqqrq gb gur yrnqreobneq. Gur uvag sbe guvf svefg cebwrpg, juvpu znl pbagnva nantenzzrq jbeqf, vf: Jbys bs ntragvir cnegvpyrf gung qvfcebir terral gb lbhe ubzrf qan gung cebjfr lbhe fgbby