```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>                    MAIN.C
#include <time.h>                      RELATIVO A PENTA.C
#include "penta.h"

typedef struct list {
        int x;
        struct list* tail;
} Str;


// alloca una struct list contenente x e tail e ne restituisce il puntatore in memoria
struct list* construct(int x, struct list* tail) {

        Str* list = malloc(sizeof(Str));
        if (!list) {
                printf("memory allocation error for list.\n");
                return NULL;
        }
        list->x = x; // vistoche è un int non uso la strncpy che è per le stringhe
        list->tail = tail; // ogni nuovo nodo creato con malloc punta al nodo precedente grazie a tail

        return list;
}


// determina se la lista l contiene almeno un elemento x pentafratto
// (si chiami la funzione opportuna dell'Esercizio 1)
int at_least_one(struct list* l) {

        int how_many = 0;
        for (; l != NULL; l = l->tail) {
                if (is_pentafract(l->x) == 1) {
                        how_many++;
                        return (l->x);
                }
        }
        if (how_many == 0) {
                printf("\nno pentafract found.");
                return 0;
        }
}

/*
main.c deve contenere una funzione iniziale main che esegue le seguenti operazioni:
1. crea una lista 5 → 12 → 6 → 15;
2. chiama at least one passando tale lista come parametro;
3. stampa il valore ritornato dalla chiamata di funzione del punto precedente
*/
```

```
int main(void) {

	struct list* head = construct(0, NULL);
	int x_value;
	int nv;
	int counter = 0;
	int choice = 2;

	printf("\nmanual(1) or auto(0) fill?\nchoice: ");
	scanf("%i", &choice);
	if (choice == 1) {
		printf("\nhow many values have to be added to the list?\nlength: ");
		scanf("%i", &nv);
		if (nv < 0) {
			printf("\ninvalid input, retry. nv must be bigger than 0");
			do {
				if (counter == 5) {
					printf("\nexceeded max number of attempts, terminating.\n");
					return 0;
				}
				printf("\n");
				printf("\ninsert list length: ");
				scanf("%i", &nv);
				counter++;
			} while (nv < 0);
		}

		for (int temp = 0; temp < nv; temp++) {
			printf("\ninsert the %d number of the list: ", temp + 1);
			scanf("%i", &x_value);
			printf("\nthe value is: %d", x_value);
			head = construct(x_value, head);
		}
	}

	else if (choice == 0) {
		head = construct(15, head);
		head = construct(6, head);
		head = construct(12, head);
		head = construct(5, head);
	}

	else {
		printf("\ninvalid input, terminating...");
		return 0;
	}
	int result = at_least_one(head);
	printf("returned value is: %d.\n", result);

	printf("\n");
	return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>                                PENTA.C
#include <time.h>
#include <math.h>
#include "penta.h"

// stampa su un’unica riga il contenuto dell’array arr, lungo length, poi va a capo
void print(int arr[], int length) {
        printf("\n");
        for (int boop = 0; boop < length; boop++) {
                printf("%d ", arr[boop]);
        }
        printf("\n");
}
//                      - COMPLETE -

// determina se il numero n non negativo e’ pentafratto,
// cioe’ se ha almeno 5 divisori interi positivi
int is_pentafract(int n) {
        int ntc = sqrt(n);              // per trovare i divisori basta arrrivare alla radice del numero?
        int counter = 0;
        for (int boopie = 1; boopie <= ntc; boopie++) { //nel caso ntc sia un quadrato perfetto
                if (n % boopie == 0) { // n diviso boopie da resto = 0
                        counter++;
                        if (boopie != n / boopie) {
                                counter++;
                        }
                }
                if (counter >= 5) {
                        return 1;
                }
        }
        return 0;
}
//                      - COMPLETE -

// modifica l’array, lungo length, in modo da spostare al suo inizio i suoi elementi
// pentafratti e alla sua fine i suoi elementi non pentafratti
void pentafract_first(int arr[], int length) {
        int index = 0;
        for (int nooty = 0; nooty < length; nooty++) {
                if (is_pentafract(arr[nooty]) == 1) {
                        print(arr, length);
                        printf("\n'%d' has to be moved.", arr[nooty]);
                        int temp = arr[index];
                        arr[index] = arr[nooty];
                        arr[nooty] = temp;
                        index++;
                }
        }
}
```

```
// inizializza arr, lungo length, con numeri interi casuali tra 0 a 999,
// usando srand() e rand()
void init_random(int arr[], int length) {
	srand(time(NULL));
	for (int squishy = 0; squishy < length; squishy++) {

		arr[squishy] = rand() % 1000;
	}
	// riempie arr
	print(arr, length);
	// stampa arr
	pentafract_first(arr, length);
	// riordina
	print(arr, length);
}
//			- COMPLETE -


int main(void) {
	int length;
	int counter = 5;

	printf("\nhow long do you want the array to be?\nlength: ");
	scanf("%i", &length);
	if (length < 0) {
		do {
			counter--;
			printf("\ninvalid input. length must be > 0 and you put '%d'.\n retry: ", length);
			scanf("%i", &length);
			if (counter == 0) {
				printf("\nexeeded max number of attempts, we'll use a predeterminated value.");
				length = 10;
				break;
			}
		} while (length < 0);
	}

	int arr[length];
	init_random(arr, length);

	printf("\n");
	return 0;
}
//			- COMPLETE -
```

```
#ifndef PENTA_H
#define PENTA_H
```

PENTA.H

```
int is_pentafract(int n);

#endif
```

INSERTION SORT

```
// Funzione per ordinare l'array con insertion sort
void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        // Spostiamo gli elementi di arr[0..i-1] che sono più grandi di key
        // verso una posizione avanti di una
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

// Funzione per stampare l'array
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Array originale: \n");
    printArray(arr, n);

    insertionSort(arr, n);

    printf("Array ordinato: \n");
    printArray(arr, n);

    return 0;
}
```

START A STRUCT

```
// Definizione della struct
struct Persona {
    char nome[50];
    int eta;
    float altezza;
};

int main() {
    // Inizializzazione parziale della struct
    struct Persona p1 = {"Alice", 25}; // altezza sarà 0.0

    printf("Nome: %s\n", p1.nome);
    printf("Età: %d\n", p1.eta);
    printf("Altezza: %.2f\n", p1.altezza);

    return 0;
}
```

```
// Definizione della struct
typedef struct list {
   int x;                                   CREA LISTA
   struct list* next;                       ADD HEAD || TAIL
} List;

// Funzione per aggiungere un elemento in testa
void aggiungiInTesta(List** head, int x) {
   List* nuovoNodo = malloc(sizeof(List));
   nuovoNodo->x = x;
   nuovoNodo->next = *head;
   *head = nuovoNodo;
}

// Funzione per aggiungere un elemento in coda
void aggiungiInCoda(List** head, int x) {
   List* nuovoNodo = malloc(sizeof(List));
   nuovoNodo->x = x;
   nuovoNodo->next = NULL;

   if (*head == NULL) {
      *head = nuovoNodo;
   } else {
      List* temp = *head;
      while (temp->next != NULL) {
         temp = temp->next;
      }
      temp->next = nuovoNodo;
   }
}

// Funzione per stampare la lista
void stampaLista(List* head) {
   while (head != NULL) {
      printf("%d -> ", head->x);
      head = head->next;
   }
   printf("NULL\n");
}

// Funzione per liberare la lista
void liberaLista(List* head) {
        if (head == NULL) {
                return;                 // Base case: lista vuota, niente da fare
        }                               // Chiamata ricorsiva per liberare i nodi successivi
        liberaLista(head->next);        // Libera la memoria del nodo corrente
        free(head);
}
```

IS PRIME?

```
int is_prime(int num) {
    if (num <= 1) {
        return 0;  // I numeri minori o uguali a 1 non sono primi
    }

    for (int i = 2; i * i <= num; i++) {  // Verifica fino alla radice quadrata di num
        if (num % i == 0) {
            return 0;  // Se num è divisibile per i, non è primo
        }
    }

    return 1;  // Se non trova divisori, il numero è primo
}

int main() {
    int num;
    printf("Inserisci un numero: ");
    scanf("%d", &num);

    if (is_prime(num)) {
        printf("%d è un numero primo.\n", num);
    } else {
        printf("%d non è un numero primo.\n", num);
    }

    return 0;
}
```

```c
#include <stdio.h>
#include <string.h>
```

STRCPY && STRCMP

```c
int main() {
    char str1[] = "Ciao";
    char str2[] = "Ciao!";
    char str_copy[50];  // Assicurati che ci sia abbastanza spazio per copiare la stringa

    // Copia la stringa str1 in str_copy
    strcpy(str_copy, str1);
    printf("La stringa copiata è: %s\n", str_copy);

    // Confronta str1 con str2
    int result = strcmp(str1, str2);
    if (result == 0) {
        printf("Le due stringhe sono uguali.\n");
    } else if (result < 0) {
        printf("La prima stringa è minore della seconda.\n");
    } else {
        printf("La prima stringa è maggiore della seconda.\n");
    }

    return 0;
}
```

SOMMA ELEMENTI ARRAY

```c
int somma_array(int arr[], int length) {
        if (length == 0)  {// Caso base
                return 0;
        }
        return arr[0] + somma_array(arr + 1, length - 1);  // Passo ricorsivo
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int length = sizeof(arr) / sizeof(arr[0]);
    printf("Somma: %d\n", somma_array(arr, length));
    return 0;
}
```

Code - Char			
0 - NULL(Null)	32 - SPACE	64 - @	96 - `
1 - SOH(Start of Heading)	33 - !	65 - A	97 - a
2 - STX(Start of Text)	34 - "	66 - B	98 - b
3 - ETX (End of Text)	35 - #	67 - C	99 - c
4 - EOT (End of Transmission)	36 - $	68 - D	100 - d
5 - ENQ (Enquiry)	37 - %	69 - E	101 - e
6 - ACK (Acknowledge)	38 - &	70 - F	102 - f
7 - BEL (Bell)	39 - '	71 - G	103 - g
8 - BS (BackSpace)	40 - (	72 - H	104 - h
9 - TAB(Horizontal Tab)	41 -)	73 - I	105 - i
10 - LF(Line Feed)	42 - *	74 - J	106 - j
11 - VT(Vertical Tabulation)	43 - +	75 - K	107 - k
12 - FF(Form Feed)	44 - ,	76 - L	108 - l
13 - CR(Carriage Return)	45 - -	77 - M	109 - m
14 - SO(Shift Out)	46 - .	78 - N	110 - n
15 - SI(Shift In)	47 - /	79 - O	111 - o
16 - DLE(Data Link Escape)	48 - 0	80 - P	112 - p
17 - C4 (Device Control 1)	49 - 1	81 - Q	113 - q
18 - DC2(Device Control 2)	50 - 2	82 - R	114 - r
19 - DC3(Device Control 3)	51 - 3	83 - S	115 - s
20 - DCA (Device Control 4)	52 - 4	84 - T	116 - t
21 - NAK(Negative ack)	53 - 5	85 - U	117 - u
22 - SYN(Synchronous Idle)	54 - 6	86 - V	118 - v
23 - ETB(End of Transmission)	55 - 7	87 - W	119 - w
24 - CAN(Cancel)	56 - 8	88 - X	120 - x
25 - EM(End of Medium)	57 - 9	89 - Y	121 - y
26 - SUB(Substitute)	58 - :	90 - Z	122 - z
27 - ESC(Escape)	59 - ;	91 - [	123 - {
28 - FS(File Separator)	60 - <	92 - \	124 - |
29 - GS(Group Separator)	61 - =	93 -]	125 - }
30 - RS(Record Separator)	62 - >	94 - ^	126 - ~
31 - US(Unit Separator)	63 - ?	95 - _	127 - DEL

ARRAY RANDOM
SINE RIPETIZIONI

```
void init(char arr[], int length) {
    // Inizializza il generatore di numeri casuali
    srand(time(NULL));

    // Assicurati che l'array abbia una lunghezza sufficiente per contenere una stringa di lunghezza length-1
    if (length <= 1) {
        arr[0] = '\0';  // Stringa vuota per lunghezza non valida
        return;
    }

    // Inizializza il primo carattere
    arr[0] = '0' + rand() % 10;  // Assegna un numero casuale da '0' a '9'

    // Inizializza gli altri caratteri
    for (int i = 1; i < length - 1; i++) {
        char next_char;
        do {
            next_char = '0' + rand() % 10;  // Genera un carattere casuale
        } while (next_char == arr[i - 1]);                          // NEXT_CHAR || ARR[I -1]

        arr[i] = next_char;
    }

    // Termina la stringa con il carattere nullo
    arr[length - 1] = '\0';
}
```

FIBONACCI

```
int fibonacci(int n) {
   if (n <= 1)
      return n;
   return fibonacci(n - 1) + fibonacci(n - 2);
}
```

FACTORIAL

```
int fattoriale(int n) {
   if (n == 0)
      return 1;
   return n * fattoriale(n - 1);
}
```

---

gcc main.c -o main → standard

gcc main..c -o main -lm → se usa math.h

gcc main.c file.c -o programma → se usa file.h

librerie

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <string.h>
#include <time.h>
#include <math.h>
```

ESAME 5 FEBBRAIO

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"

struct List *create(int head, struct List *tail) {
  struct List *this = (struct List *) malloc(sizeof(struct List));
  this->head = head;
  this->tail = tail;
  return this;
}

void print(struct List *this) {
  if (this == NULL)
    printf("\n");
  else {
    printf("%i ", this->head);
    print(this->tail);
  }
}

struct List *from(int arr[], int length) {
  struct List *result = NULL;

  for (int pos = length - 1; pos >= 0; pos--)
    result = create(arr[pos], result);

  return result;
}

struct List *filter(struct List *this, int threshold) {
  if (this == NULL)
    return NULL;
  else if (this->head >= threshold)
    return create(this->head, filter(this->tail, threshold));
  else
    return filter(this->tail, threshold);
}

struct List *duplicate(struct List *this) {
  if (this == NULL)
    return NULL;
  else
    return create(this->head, create(this->head, duplicate(this->tail)));
}
```