Esami Svolti Parte a Crocette – Programmazione 2 (C)

Autore: **Scarlata Davide**
Anno: **Primo anno**
Corso: **Linguaggio C – Programmazione 2**

> ⚠️ **Nota bene:**
> Le soluzioni non sono garantite corrette: **sono esercizi svolti da me** (studente) e **potrebbero contenere errori**.
> Questo materiale è pensato come supporto allo studio e confronto personale, **non come riferimento ufficiale**.

🔗 Collegamenti utili

- 🌐 Moodle corso C
- 📝 Piattaforma esami

📘 Come usare questa raccolta

- Puoi usare questi esercizi per **ripassare la teoria** attraverso il codice.
- Prova a **risolvere prima da solo** ogni esercizio, poi confronta con la mia soluzione.
- Se trovi un errore o vuoi suggerire una miglioria, **scrivimi** e ti fornirò il codice Markdown per effettuare un *push* della modifica.

🙏 Ringraziamenti

Un ringraziamento speciale a **edo.js** e a chi ha collaborato alla creazione della **"Bibbia" di Programmazione 2**:
una risorsa fondamentale per comprendere e affrontare al meglio questo corso.

☕ Se vuoi offrirmi un caffè

- paypal
- revolut

Primo Appello

```c
typedef struct _scheda Scheda, *Puntatore;

struct _scheda {
    int numero;
    char *testo;
};

int funzione(Scheda sc, Puntatore pu);

Scheda x;
Scheda *y;
Puntatore *z;
```

Risposte

Invocazione	Corretta o Errata
`funzione(x, NULL);`	CORRETTA
`funzione(**z, y);`	CORRETTA
`funzione(y, "y");`	ERRATA

Invocazioni da verificare:

1. `funzione(x, NULL);`

- `x` è `Scheda` → ok per primo argomento
  `NULL` è un puntatore nullo → ok per secondo argomento (Puntatore)
  Quindi: CORRETTA

2. `funzione(**z, y);`

- `z` è `Puntatore *` = `Scheda **`

- `*z` è `Puntatore` = `Scheda *`
- `**z` è `Scheda` (dereferenzia due volte)
  Primo argomento: `**z` → `Scheda` → OK.
  Secondo argomento: `y` è `Scheda *` = `Puntatore` → OK.
  Quindi: CORRETTA

2. `funzione(y, "y");`

- Primo argomento: `y` è `Scheda *` (puntatore), mentre la funzione vuole `Scheda` (struct per valore) → NON compatibile.
- Secondo argomento: `"y"` è `const char *`, mentre la funzione vuole `Puntatore` (`Scheda *`) → NON compatibile.
  Quindi: ERRATA

```
int arr[] = {1, 4,-1, 5}
int funz(int *a, int n) {
    if (a-- NULL) return -1;
    if (n 0) return -2;
    int *ptraj
    int i 0;
    int sum 0;
    while (i < n-1) {
    }
    if (a[i]%2 == 0) {
        i++;
        sum sum *(ptr+i);
    }
    else {
        sum = sum + ptr[i+1];
        i++;
    }
    return sum
}
```

Affermazione	Risposta
La funzione può produrre segmentation fault?	VERO (potenzialmente)
Se non produce segmentation fault, terminerà sempre?	VERO
funz(arr,4) restituisce:	**13**

Può produrre segmentation fault?

Vero.

- Se `a` è `NULL` (ma la funzione controlla e ritorna -1 in questo caso, quindi no).
- Il problema è con l'accesso a `ptr[i + 1]` quando `i` è all'ultimo elemento.
- La condizione del ciclo è `i < n - 1`, quindi `i + 1` arriva fino a `n - 1`, che è valido.
- Però se dentro `else` usiamo `ptr[i + 1]`, ci può essere un problema se `i + 1` supera la dimensione dell'array, ad esempio se `n` è piccolo.

Se non produce segmentation fault, terminerà sempre?

Vero.

- Il ciclo ha una condizione ben definita e `i` viene sempre incrementato.
- Non ci sono loop infiniti o condizioni che impediscano l'uscita.
- La funzione termina sempre con un `return sum`.

Calcolo `funz(arr, 4)`

Facciamo un passo alla volta:

- `arr = {1, 4, -1, 5}`
- Ciclo `while (i < 3)`:

i	a[i]	a[i]%2==0?	Operazione	sum aggiornato	i dopo
0	1	no (1%2=1)	sum += ptr[i+1]=4	sum = 0 + 4 =4	i=1
1	4	sì (4%2=0)	sum += ptr[i]=4	sum = 4 + 4=8	i=2
2	-1	no (-1%2=-1)	sum += ptr[i+1]=5	sum = 8 + 5=13	i=3

Termina il ciclo perché `i == 3` non è < 3.
La funzione ritorna `sum = 13`.

```
typedef struct _indUK {
    int number;
    char street[N1];
    int floor
    char pcode[N2];
    char fown[N3];
} indUK;

enum tag strada { VIA, PIAZZA, CORSO];
```

```c
typedef struct _indITA {
    union {
    } n;
    indUK address;
    indITA indirizzo;
} X, X2, *PTRX;

int civico;
enum tag strada strada;
char nomestrada[N1]
char citta[N3];
} indITA;

struct {
    enum tag_paese nazione;
    union {
    } n;
    indUK address;
    indITA indirizzo;
} X, X2, *PTRX;
```

Espressione	Corretta o Errata	Motivo principale
`X.n.address.civico = 10;`	Errata	`X` è un tipo, non una variabile
`if (nazione == UK) strcpy(X.n.address.street, "Magellan Lane");`	Errata	`X` è un tipo, non una variabile
`PTRX = &X;`	corretta	
`PTRX = &(X->nazione);`	Errata	`X` è tipo, `->` non valido su tipo, tipi incompatibili
`PTRX->n.indirizzo.nomestrada[0] == 'z';`	corretta	
`PTRX->nomestrada`	Errata	`PTRX` è tipo, campo non diretto della struct

Secondo Appello

```c
typedef struct _oggetto {
    char* descrizione;
    char codice[10];
    int peso;
} Oggetto, *Puntatore;
```

```
int funzione(Oggetto og, Puntatore pu);

Oggetto scatola[5];
Oggetto *x;
Puntatore *z;
```

Chiamata	Corretta o Errata	Motivazione
`funzione(x, NULL);`	Errata	Primo argomento deve essere `Oggetto`, non `Oggetto *`
`funzione(**z, z);`	Errata	Secondo argomento deve essere `Oggetto *`, non `Oggetto **`
`funzione(*z, &(scatola[3]));`	Errata	Primo argomento deve essere `Oggetto`, non `Oggetto *`

```
char* arr = "ELEFANTE";

int funz(char *a, int n, char c) {
    if (a == NULL) return -1;
    if (n < 0) return -2;
    int i = 0;
    int sum = 0;
    while ((i < 5) || (a[i] != '\0')) {
        if (a[i] != c) {
            i = i + 2;
            sum = sum + 1;
        }
        else {
            sum = sum - 1;
            i = i - 1;
        }
    }
    return sum;
}
```

Domanda	Risposta	Motivazione
È possibile che qualche esecuzione di `funz` produca un segmentation fault?	VERO	`i` può diventare negativo o superare la lunghezza della stringa, causando accesso a memoria invalida

Domanda	Risposta	Motivazione
La chiamata di funzione funz(arr, 5, 'A') restituisce (scegliere un valore numerico fra quelli proposti nel menu, oppure VERO se l'esecuzione può non terminare o produrre un segmentation fault)	VERO	
È possibile che qualche esecuzione di `funz` non termini?	VERO	Il ciclo può diventare infinito se `i` continua a oscillare senza mai uscire dalla condizione

```c
enum tag_ristorante {ENGLISH, ITALIANO};

typedef struct_course {
    int number;
    int price;
    char name[N1];
    char *descriprtion;
} Course;

enum tag_portata {ANTIPASTO, PRIMO, SECONDO};

typedef struct_portata {
    int prezzo;
    enum tag_portata tipo;
    char nomeportata[N1];
} Portata;

struct {
    enum tag_ristorante tipo;
    union {
        Course co;
        Portata po;
    } piatto;
} X, X2
```

Espressione	Corretta o Errata	Motivazione
`X.piatto.po.nomeportata != X2.piatto.po.nomeportata;`	CORRETTA	Confronto tra array decaduti a puntatori, confronto valido ma confronta indirizzi (non stringhe)

Espressione	Corretta o Errata	Motivazione
`X.piatto.po.nomeportata = X2.piatto.po.nomeportata;`	ERRATA	Non si può assegnare direttamente array in C (serve `strcpy`)
`X.piatto.po.tag_portata = "PRIMO";`	ERRATA	`tag_portata` è un enum, non una stringa. Assegnare stringa a enum non è valido
`PTRX->piatto.co.number = X.piatto.po.prezzo;`	CORRETTA	Assegnazione tra due int, tipi compatibili
`PTRX = malloc(sizeof(Portata));`	CORRETTA	Allocazione dinamica, tipo `void*` convertito implicitamente a `Menu*`
`X->piatto.co.name = PTRX.piatto.po.nomeportata;`	ERRATA	`X` non è un puntatore quindi `->` non valido; `nomeportata` è un array, non si può assegnare direttamente

Terzo Appello

```c
int arr[] = {1, 4, -1, 5};

int funz(int *a, int n) {
    if (a == NULL) return -1;
    if (n < 0) return -2;
    int *ptr = a;
    int i = 0;
    int sum = 0;
    while (i < n - 1) {
        if (a[i] % 2 == 0) {
            i++;
            sum = sum + *(ptr + i);
        } else {
            sum = sum + ptr[i + 1];
            i++;
        }
    }
    return sum;
}
```

Domanda	Risposta	Motivazione
Può produrre segmentation fault?	SÌ	Quando `i` è vicino a `n-1`, l'accesso a `ptr[i+1]` può superare i limiti dell'array, causando accesso illegale
Termina sempre se non produce segfault?	SÌ	`i` viene incrementato ad ogni iterazione, quindi il ciclo termina sicuramente se non si accede fuori limite
Valore restituito da `funz(arr, 4)`	8	Somma calcolata passo passo con input specifico, senza accessi fuori limite

```c
typedef struct scheda Scheda, *Puntatore;

struct scheda {
    int numero;
    char *testo;
};

int funzione(Scheda sc, Puntatore pu);

Scheda x;      // variabile struct scheda
Scheda *y;     // puntatore a struct scheda
Puntatore *z;  // Puntatore è già *Scheda, quindi Puntatore* è un doppio puntatore a Scheda
```

Invocazione	Corretta o Errata	Motivazione
`funzione(*y, &(**z));`	ERRATA	`&(**z)` è `Puntatore *` (cioè `Scheda **`), mentre la funzione aspetta `Puntatore` (`Scheda *`)
`funzione(y, &y);`	ERRATA	`y` è `Scheda *`, ma la funzione aspetta un `Scheda` (non un puntatore) come primo argomento
`funzione((**z), &x);`	CORRETTA	`(**z)` è `Scheda` (primo argomento corretto), `&x` è `Puntatore` (`Scheda *`), argomento corretto

```c
enum tag_paese { UK, ITALIA };

typedef struct _indUK {
    int number;
```

```
    char street[N1];
    int floor;
    char pcode[N2];
    char town[N3];
} indUK;

enum tag_strada { VIA, PIAZZA, CORSO };

typedef struct _indITA {
    int civico;
    enum tag_strada strada;
    char nomestrada[N1];
    char citta[N3];
} indITA;

struct {
    enum tag_paese nazione;
    union {
        // union vuota, da completare
    } n;
    indUK address;
    indITA indirizzo;
} X, X2, *PTRX;
```

Espressione	Corretta o Errata	Motivazione
`PTRX = malloc(sizeof(indITA));`	ERRATA	`PTRX` punta alla struttura anonima, non a `indITA`; la dimensione è sbagliata
`X->n->address->pcode = PTRX.n.address.pcode;`	ERRATA	Sintassi errata: `X` è variabile, non puntatore; `n` è union, non puntatore; `address` è struct
`PTRX->indirizzo->civico = X.n.address.number;`	ERRATA	`indirizzo` è struttura, non puntatore; uso `->` sbagliato; `X.n` è union (non puntatore)
`X.n.indirizzo.citta = X2.n.indirizzo.citta;`	ERRATA	Assegnazione diretta tra array di char non valida, serve `strcpy`
`strcpy(X2.n.indirizzo.città, PTRX->n.address.town);`	CORRETTA	Copia stringhe tra array, sintassi corretta
`if (strcmp(X2.n.address.street, X.n.indirizzo.nomestrada));`	CORRETTA (sintassi)	`strcmp` confronta stringhe, uso corretto, ma la condizione va usata in modo significativo

Primo appello 2025

```
//Si considerino le seguenti dichiarazioni di tipi e di variabili:

typedef struct portachiavi Portachiavi, *Puntatore;

struct portachiavi {

    char* via;

    int civico;

    char* chiavi[8];

};
Portachiavi port;

Puntatore pt;

//Quali delle seguenti linee di codice sono corrette e quali sono errate?
```

Linea	Codice	Corretta?	Motivo
1	`pt = (Portachiavi)malloc(sizeof(Portachiavi));`	❌ ERRATA	Cast sbagliato a tipo non puntatore
2	`port = (Portachiavi)malloc(sizeof(struct portachiavi));`	❌ ERRATA	Assegna un puntatore a una struttura
3	`pt = (Puntatore)malloc(sizeof(Portachiavi)*10);`	✅ CORRETTA	Allocazione dinamica corretta
4	`port = (Portachiavi)malloc(sizeof(port)*10);`	❌ ERRATA	Cast e tipo errato nel lato sinistro
5	`(pt = &port)--;`	❌ ERRATA	Sintassi non valida / non sicura

```
//Si consideri il codice riportato nel seguito e, senza fare supposizioni su parti di codice non riportate, si risponda alle domande:
```

```c
int foo(char *s) {

   int cont = 0;

   if (*s == 'E') { cont--; }

   else if (*s == 'R') { cont = foo(s+1); }

   else if (*s == 'Q') { cont = foo(s-1) + 1; }

   else cont = cont +  foo(s+2);

  return cont;
}
```

Domanda	Risposta	Spiegazione breve
1. La funzione può non terminare	✅ VERO	Nessun caso base presente
2. Il parametro `s` è una stringa	❌ FALSO	È solo un `char *`, non necessariamente una stringa
3. La funzione può produrre un segmentation fault	✅ VERO	Accessi a `s-1` o `s+N` possono uscire dai limiti
4. `foo("CORRETTO")` restituisce	✅ **-1**	Tracciamento mostra decremento alla fine

```c
//Si considerino le seguenti dichiarazioni di tipi e di variabili:
typedef struct astuccio Astuccio, *Puntatore;
struct astuccio {
char* colore;
char* matite[10];
int prezzo;
};
Astuccio ast;
Puntatore pun;
void *genp;
//Quali delle seguenti linee di codice sono corrette e quali sono errate?
```

Riga	Corretta?	Commento
`pun = (Astuccio)malloc(sizeof(Astuccio));`	❌ FALSO	Cast sbagliato: `Astuccio` è una struct, non un puntatore

Riga	Corretta?	Commento
`ast = (astuccio)malloc(sizeof(astuccio));`	❌FALSO	Tipi errati: `astuccio` non è definito come tipo
`pun = (Puntatore)malloc(sizeof(Astuccio)*5);`	✅ VERO	Corretto
`ast = (Astuccio)malloc(sizeof(ast)*5);`	❌ FALSO	Assegnazione di puntatore a struct
`(pun = &ast)++;`	❌ FALSO	Sintatticamente valido, ma semantica errata
`genp = malloc(sizeof(Astuccio));`	✅ VERO	Corretto in C (void* può essere assegnato senza cast)

```c
int funz(char *a) {
int cont = 0;
if (*a == 'E') { cont-; }
else if (*a == 'R') { cont = funz(a+1); }
else if (*a == 'Q') { cont = funz(a-1) + 1; }
else cont = cont + funz(a+2);
return cont;
}
```

#	Affermazione	Corretta?	Spiegazione
1	La funzione può non terminare	✅ VERO	In mancanza di controllo su `\0`, può diventare ricorsiva infinita
2	Il parametro `a` è una stringa	❌ FALSO	È un `char*`, ma non si può assumere che punti a una stringa terminata
3	La funzione può produrre un segmentation fault	✅ VERO	Possibile con `funz(a - 1)` se `a` è al primo elemento
4	La chiamata `funz("CARRETTO")` restituisce (−1)	✅ VERO	Tracciamento conferma che il risultato è -1