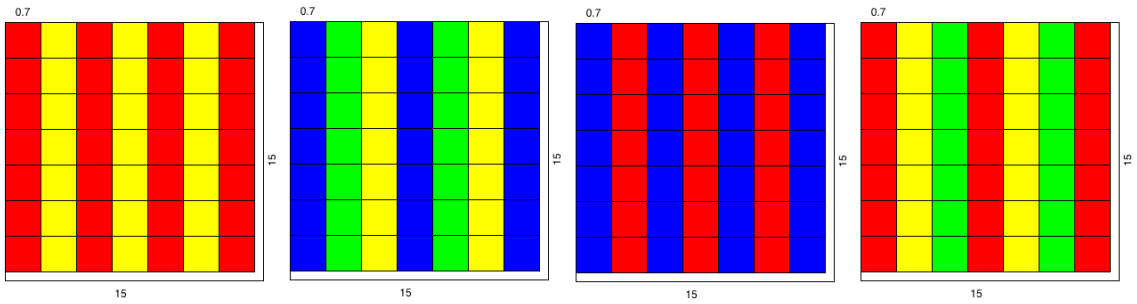
Esame 20260108

Esercizio 2

(1) **Esercizio 2 v1**

ESSAY | marked out of 10 | penalty 0 | File picker

Un tecnico edile deve calcolare e disegnare il pavimento di una stanza rettangolare, utilizzando mattonelle quadrate di lato noto. Per fare ciò, ha bisogno di un programma che, data la lunghezza e la larghezza della stanza (in metri), e la dimensione del lato delle mattonelle quadrate, calcoli quante mattonelle sono necessarie per coprire l'intera superficie della stanza. Il pavimento deve essere coperto con mattonelle intere, senza tagliarle, e le mattonelle in eccesso (se presenti) devono essere calcolate come residuo. Il pavimento é rappresentato dalla seguente struttura `Pavimento` con due campi reali `lunghezza` e `larghezza`, un campo `area` che rapresenta come un array di liste concatenate terminate da `nullptr` dove ogni nodo della lista contiene un `colore` che indica il colore della rispettiva mattonella, e un puntatore al prossimo elemento. Per coprire il pavimento, le mattonelle devono essere disposte in modo alternato (ad esempio bianco e nero), come in una scacchiera, e il colore della prima mattonella di ogni riga deve essere uguale al colore dell'ultima mattonella, come illustrato nella figura seguente.



Il `main` del programma é già implementato e non deve essere modificato, e chiama la funzione `CalcolaPavimento` **(da definire)** che prende come argomento la struttura `Pavimento p` (prestare attenzione a come passare il parametro), e la dimensione del lato di ogni mattonella (`double`). La funzione `CalcolaPavimento` alloca e popola l'array `p.area` che contiene le mattonelle da mettere nel pavimento `p`. Ogni elemento dell'array `p.area[i]` rappresenta la riga `i` del pavimento, e ogni nodo della lista collegata rappresenta una mattonella di quella riga, con il suo colore. Non ci deve essere memoria allocata inutilmente: se una riga del pavimento non richiede mattonelle, l'elemento corrispondente dell'array `p.area` deve essere `nullptr`. Non ci deve essere condivisione di nodi tra le liste collegate delle diverse righe del pavimento: ogni mattonella deve essere rappresentata da un nodo distinto.

la funzione `CalcolaPavimento` deve creare il pavimento in modo che le mattonelle di ogni riga usino alternati i colori `Rosso` e `Giallo`, inizino con il colore `Rosso` e terminano con il colore `Rosso` (si veda primo pattern a sinistra nella figura sopra).

La funzione `CalcolaPavimento` **deve essere ricorsiva** e **NON deve contenere iteratori** espliciti (`for`, `while`, `do-while`). La funzione `CalcolaPavimento` può ovviamente contenere codice sequenziale o condizionale. Sono consentite (se ritenute necessarie) chiamate a funzioni ricorsive ausiliarie che a loro volta **non contengano iterazioni esplicite** (`for`, `while`, `do-while`).

Il file `esercizio2.cpp` contiene tutto quanto necessario tranne la dichiarazione e la definizione della procedura `CalcolaPavimento`.

Di seguito é riportato un esempio di esecuzione del programma.

```
computer > ./a.out 12 15 3
RYRYR 0
RYRYR 0
RYRYR 0
RYRYR 0
0
computer > ./a.out 11 14 3
RYR 5
RYR 5
RYR 5
2
computer > ./a.out 11 14 2.4
RYRYR 2
RYRYR 2
RYRYR 2
RYRYR 2
1.4
computer > ./a.out 2 2 2.4
2
```

Note:

- Scaricare i file `esercizio2.cpp`, modificare il solo file `esercizio2.cpp` per inserire il codice necessario per rispondere a questo esercizio. **Caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream` e `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).
- Si ricorda che la soluzione deve essere implementata in C++ **NON usando altri elementi della C++ standard template library tranne le funzioni definite nelle librerie concesse, anche se il file compila senza cambiare gli header**!

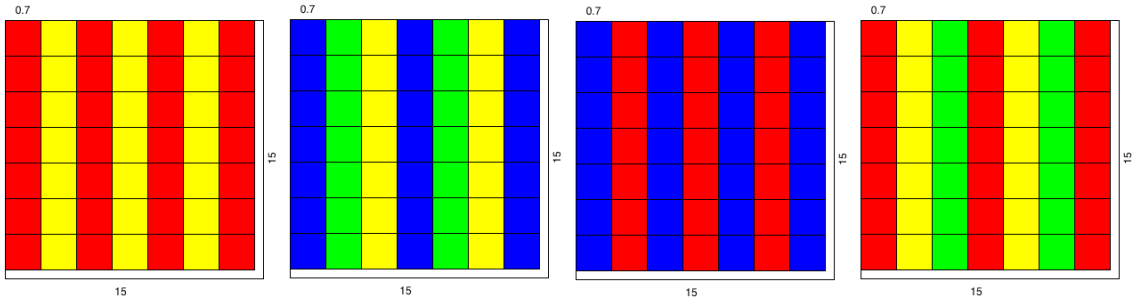
esercizio2.cpp

Information for graders:

(2) **Esercizio 2 v2**

ESSAY | marked out of 10 | penalty 0 | File picker

Un tecnico edile deve calcolare e disegnare il pavimento di una stanza rettangolare, utilizzando mattonelle quadrate di lato noto. Per fare ciò, ha bisogno di un programma che, data la lunghezza e la larghezza della stanza (in metri), e la dimensione del lato delle mattonelle quadrate, calcoli quante mattonelle sono necessarie per coprire l'intera superficie della stanza. Il pavimento deve essere coperto con mattonelle intere, senza tagliarle, e le mattonelle in eccesso (se presenti) devono essere calcolate come residuo. Il pavimento é rappresentato dalla seguente struttura `Pavimento` con due campi reali `lunghezza` e `larghezza`, un campo `area` che rapresenta come un array di liste concatenate terminate da `nullptr` dove ogni nodo della lista contiene un `colore` che indica il colore della rispettiva mattonella, e un puntatore al prossimo elemento. Per coprire il pavimento, le mattonelle devono essere disposte in modo alternato (ad esempio bianco e nero), come in una scacchiera, e il colore della prima mattonella di ogni riga deve essere uguale al colore dell'ultima mattonella, come illustrato nella figura seguente.



Il `main` del programma é già implementato e non deve essere modificato, e chiama la funzione `CalcolaPavimento` **(da definire)** che prende come argomento la struttura `Pavimento p` (prestare attenzione a come passare il parametro), e la dimensione del lato di ogni mattonella (`double`). La funzione `CalcolaPavimento` alloca e popola l'array `p.area` che contiene le mattonelle da mettere nel pavimento `p`. Ogni elemento dell'array `p.area[i]` rappresenta la riga `i` del pavimento, e ogni nodo della lista collegata rappresenta una mattonella di quella riga, con il suo colore. Non ci deve essere memoria allocata inutilmente: se una riga del pavimento non richiede mattonelle, l'elemento corrispondente dell'array `p.area` deve essere `nullptr`. Non ci deve essere condivisione di nodi tra le liste collegate delle diverse righe del pavimento: ogni mattonella deve essere rappresentata da un nodo distinto.

la funzione `CalcolaPavimento` deve creare il pavimento in modo che le mattonelle di ogni riga usino alternati i colori `Blu`, `Verde` e `Giallo`, inizino con il colore `Blu` e terminano con il colore `Blu` (si veda secondo pattern a sinistra nella figura sopra).

La funzione `CalcolaPavimento` **deve essere ricorsiva** e **NON deve contenere iteratori** espliciti (`for`, `while`, `do-while`). La funzione `CalcolaPavimento` può ovviamente contenere codice sequenziale o condizionale. Sono consentite (se ritenute necessarie) chiamate a funzioni ricorsive ausiliarie che a loro volta **non contengano iterazioni esplicite** (`for`, `while`, `do-while`).

Il file `esercizio2.cpp` contiene tutto quanto necessario tranne la dichiarazione e la definizione della procedura `CalcolaPavimento`.

Di seguito é riportato un esempio di esecuzione del programma.

```
computer > ./a.out 12 15 3
```

```
BGYB 3
BGYB 3
BGYB 3
BGYB 3
0
computer > ./a.out 11 14 3
BGYB 2
BGYB 2
BGYB 2
2
computer > ./a.out 11 14 2.4
BGYB 4.4
BGYB 4.4
BGYB 4.4
BGYB 4.4
1.4
computer > ./a.out 2 2 2.4
2
```

Note:

- Scaricare i file `esercizio2.cpp`, modificare il solo file `esercizio2.cpp` per inserire il codice necessario per rispondere a questo esercizio. **Caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream` e `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).
- Si ricorda che la soluzione deve essere implementata in C++ **NON usando altri elementi della C++ standard template library tranne le funzioni definite nelle librerie concesse, anche se il file compila senza cambiare gli header**!

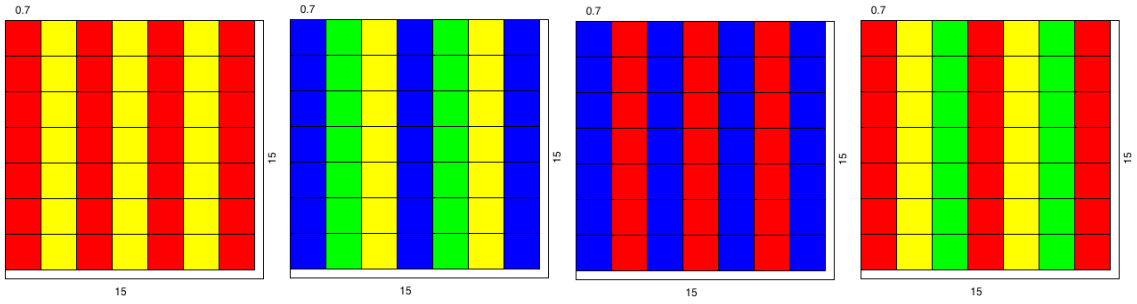
esercizio2.cpp

Information for graders:

(3) **Esercizio 2 v3**

ESSAY | marked out of 10 | penalty 0 | File picker

Un tecnico edile deve calcolare e disegnare il pavimento di una stanza rettangolare, utilizzando mattonelle quadrate di lato noto. Per fare ciò, ha bisogno di un programma che, data la lunghezza e la larghezza della stanza (in metri), e la dimensione del lato delle mattonelle quadrate, calcoli quante mattonelle sono necessarie per coprire l'intera superficie della stanza. Il pavimento deve essere coperto con mattonelle intere, senza tagliarle, e le mattonelle in eccesso (se presenti) devono essere calcolate come residuo. Il pavimento é rappresentato dalla seguente struttura `Pavimento` con due campi reali `lunghezza` e `larghezza`, un campo `area` che rapresenta come un array di liste concatenate terminate da `nullptr` dove ogni nodo della lista contiene un `colore` che indica il colore della rispettiva mattonella, e un puntatore al prossimo elemento. Per coprire il pavimento, le mattonelle devono essere disposte in modo alternato (ad esempio bianco e nero), come in una scacchiera, e il colore della prima mattonella di ogni riga deve essere uguale al colore dell'ultima mattonella, come illustrato nella figura seguente.



Il `main` del programma é già implementato e non deve essere modificato, e chiama la funzione `CalcolaPavimento` **(da definire)** che prende come argomento la struttura `Pavimento p` (prestare attenzione a come passare il parametro), e la dimensione del lato di ogni mattonella (`double`). La funzione `CalcolaPavimento` alloca e popola l'array `p.area` che contiene le mattonelle da mettere nel pavimento `p`. Ogni elemento dell'array `p.area[i]` rappresenta la riga `i` del pavimento, e ogni nodo della lista collegata rappresenta una mattonella di quella riga, con il suo colore. Non ci deve essere memoria allocata inutilmente: se una riga del pavimento non richiede mattonelle, l'elemento corrispondente dell'array `p.area` deve essere `nullptr`. Non ci deve essere condivisione di nodi tra le liste collegate delle diverse righe del pavimento: ogni mattonella deve essere rappresentata da un nodo distinto.

la funzione `CalcolaPavimento` deve creare il pavimento in modo che le mattonelle di ogni riga usino alternati i colori `Blu` e `Rosso`, inizino con il colore `Blu` e terminano con il colore `Blu` (si veda secondo pattern a destra nella figura sopra).

La funzione `CalcolaPavimento` **deve essere ricorsiva** e **NON deve contenere iteratori** espliciti (`for`, `while`, `do-while`). La funzione `CalcolaPavimento` può ovviamente contenere codice sequenziale o condizionale. Sono consentite (se ritenute necessarie) chiamate a funzioni ricorsive ausiliarie che a loro volta **non contengano iterazioni esplicite** (`for`, `while`, `do-while`).

Il file `esercizio2.cpp` contiene tutto quanto necessario tranne la dichiarazione e la definizione della procedura `CalcolaPavimento`.

Di seguito é riportato un esempio di esecuzione del programma.

```
computer > ./a.out 12 15 3
```

```
BRBRB 0
BRBRB 0
BRBRB 0
BRBRB 0
0
computer > ./a.out 11 14 3
BRB 5
BRB 5
BRB 5
2
computer > ./a.out 11 14 2.4
BRBRB 2
BRBRB 2
BRBRB 2
BRBRB 2
1.4
computer > ./a.out 2 2 2.4
2
```

Note:

- Scaricare i file `esercizio2.cpp`, modificare il solo file `esercizio2.cpp` per inserire il codice necessario per rispondere a questo esercizio. **Caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream` e `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).
- Si ricorda che la soluzione deve essere implementata in C++ **NON usando altri elementi della C++ standard template library tranne le funzioni definite nelle librerie concesse, anche se il file compila senza cambiare gli header**!

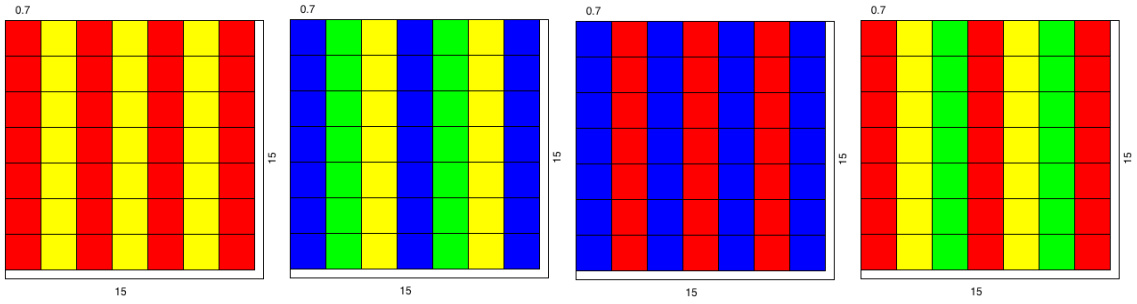
esercizio2.cpp

Information for graders:

(4) **Esercizio 2 v4**

ESSAY | marked out of 10 | penalty 0 | File picker

Un tecnico edile deve calcolare e disegnare il pavimento di una stanza rettangolare, utilizzando mattonelle quadrate di lato noto. Per fare ciò, ha bisogno di un programma che, data la lunghezza e la larghezza della stanza (in metri), e la dimensione del lato delle mattonelle quadrate, calcoli quante mattonelle sono necessarie per coprire l'intera superficie della stanza. Il pavimento deve essere coperto con mattonelle intere, senza tagliarle, e le mattonelle in eccesso (se presenti) devono essere calcolate come residuo. Il pavimento é rappresentato dalla seguente struttura `Pavimento` con due campi reali `lunghezza` e `larghezza`, un campo `area` che rapresenta come un array di liste concatenate terminate da `nullptr` dove ogni nodo della lista contiene un `colore` che indica il colore della rispettiva mattonella, e un puntatore al prossimo elemento. Per coprire il pavimento, le mattonelle devono essere disposte in modo alternato (ad esempio bianco e nero), come in una scacchiera, e il colore della prima mattonella di ogni riga deve essere uguale al colore dell'ultima mattonella, come illustrato nella figura seguente.



Il `main` del programma é già implementato e non deve essere modificato, e chiama la funzione `CalcolaPavimento` **(da definire)** che prende come argomento la struttura `Pavimento p` (prestare attenzione a come passare il parametro), e la dimensione del lato di ogni mattonella (`double`). La funzione `CalcolaPavimento` alloca e popola l'array `p.area` che contiene le mattonelle da mettere nel pavimento `p`. Ogni elemento dell'array `p.area[i]` rappresenta la riga `i` del pavimento, e ogni nodo della lista collegata rappresenta una mattonella di quella riga, con il suo colore. Non ci deve essere memoria allocata inutilmente: se una riga del pavimento non richiede mattonelle, l'elemento corrispondente dell'array `p.area` deve essere `nullptr`. Non ci deve essere condivisione di nodi tra le liste collegate delle diverse righe del pavimento: ogni mattonella deve essere rappresentata da un nodo distinto.

la funzione `CalcolaPavimento` deve creare il pavimento in modo che le mattonelle di ogni riga usino alternati i colori `Rosso`, `Giallo` e `Verde`, inizino con il colore `Rosso` e terminano con il colore `Rosso` (si veda primo pattern a destra nella figura sopra).

La funzione `CalcolaPavimento` **deve essere ricorsiva** e **NON deve contenere iteratori** espliciti (`for`, `while`, `do-while`). La funzione `CalcolaPavimento` può ovviamente contenere codice sequenziale o condizionale. Sono consentite (se ritenute necessarie) chiamate a funzioni ricorsive ausiliarie che a loro volta **non contengano iterazioni esplicite** (`for`, `while`, `do-while`).

Il file `esercizio2.cpp` contiene tutto quanto necessario tranne la dichiarazione e la definizione della procedura `CalcolaPavimento`.

Di seguito é riportato un esempio di esecuzione del programma.

```
computer > ./a.out 12 15 3
```

```
RYGR 3
RYGR 3
RYGR 3
RYGR 3
0
computer > ./a.out 11 14 3
RYGR 2
RYGR 2
RYGR 2
2
computer > ./a.out 11 14 2.4
RYGR 4.4
RYGR 4.4
RYGR 4.4
RYGR 4.4
1.4
computer > ./a.out 2 2 2.4
2
```

Note:

- Scaricare i file `esercizio2.cpp`, modificare il solo file `esercizio2.cpp` per inserire il codice necessario per rispondere a questo esercizio. **Caricare il file sorgente risultato delle vostre modifiche a soluzione di questo esercizio** nello spazio apposito.
- All'interno di questo programma **non è ammesso** l'utilizzo di variabili globali o di tipo `static` e di funzioni di libreria al di fuori di quelle definite in `iostream` e `cstdlib`.
- Si ricorda che, gli esempi di esecuzione sono puramente indicativi, e la soluzione proposta NON deve funzionare solo per l'input fornito, ma deve essere robusta a variazioni compatibili con la specifica riportata in questo testo.
- Si ricorda di inserire solo nuovo codice e di **NON MODIFICARE** il resto del programma (pena annullamento dell'esercizio).
- Si ricorda che la soluzione deve essere implementata in C++ **NON usando altri elementi della C++ standard template library tranne le funzioni definite nelle librerie concesse, anche se il file compila senza cambiare gli header**!

esercizio2.cpp

Information for graders:

Total of marks: 40