Informatica - Mod. Programmazione
Lezione 08

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica
Università di Bergamo

Allocazione Dinamica

- Come fare a gestire insiemi di dati di cui non si conosce la dimensione quando si scrive il programma? Esistono due scenari possibili
- **Scenario 1**
  Prima di raccogliere i dati, si riesce a sapere quanti dati arriveranno
- **Scenario 2**
  Non vi è modo alcuno di sapere quanti dati arriveranno

Allocazione Dinamica

- Per risolvere entrambe gli scenari, si usa l'**ALLOCAZIONE DINAMICA**
- Si richiede al sistema operativo di riservare uno spazio di memoria, di cui si ottiene l'indirizzo
  ALLOCAZIONE
- Una volta terminato l'uso di questo spazio, occorre rilasciarlo
  DE-ALLOCAZIONE

Allocazione Dinamica

In C++

- **ALLOCAZIONE**
  Istruzione `new`
- **DE-ALLOCAZIONE**
  Istruzione `delete`

Allocazione

- Dato un tipo `T`
  Allocazione di un elemento di tipo `T`
  `new T`
- l'istruzione `new` cerca di allocare lo spazio per il tipo `T`:
  - se riesce, restituisce l'indirizzo dell'elemento (`T *`)
  - se non riesce, restituisce il valore `NULL` (costante predefinita che vale `0`)
- Esempio:

```
T *p;
p = new T;
```

Allocazione di un Vettore

- Dato un tipo `T`
  Allocazione di un vettore di 5 elementi di tipo `T`
  `new T[5]`
- l'istruzione `new` cerca di allocare lo spazio per il vettore di tipo `T`:
  - se riesce, restituisce l'indirizzo del primo elemento (`T *`)
  - se non riesce, restituisce il valore `NULL`
- Esempio:

```
T *v;
v = new T[5];
```

De-allocazione

- Dato un puntatore `p` che punta ad un tipo `T`, ottenuto per allocazione dinamica

  ```
  delete p;
  ```

- Lo spazio di memoria puntato viene de-allocato e reso disponibile per utilizzi futuri

Allocazione Dinamica

Heap

- Dove viene allocato lo spazio di memoria?
- Nello **HEAP** (mucchio)
- Si tarata di uno spazio di memoria distinto dallo stack delle chiamate

Programma: Dinamica_01.cpp

```
int main()
{
    int i;
    int *elenco;
    int size;

    cout << "Quanti Valori? ";
    cin >> size;

    elenco = new int[size];
    if(elenco == NULL)
    {
        cout << "Memoria Esaurita";
        exit(1);
    }
```

Programma: Dinamica_01.cpp

```
    for(i=0; i < size; i++)
    {
        cout << "Valore " << (i+1) << ": ";
        cin >> elenco[i];
    }

    for(i=0; i < size; i++)
        cout << "Valore " << (i+1) << ": "
             << elenco[i] << endl;

    delete elenco;

    return 0;
}
```

Allocazione Dinamica

- Che fare se l'allocazione non riesce?
- Soluzione semplice:
  si termina il programma con una condizione di errore
- Potremmo scrivere
  `return 1;`
- Ma se non siamo nella funzione `main`, dobbiamo usare un'altra soluzione:
  `exit(1);`

Funzione `exit`

- Era già presente nella libreria del C
  in `stdlib.h`
- In C++, la libreria equivalente si chiama
  `cstdlib`
- La funzione interrompe l'esecuzione del processo.
  Il valore tra parentesi viene restituito al sistema
  operativo

Programma Ordinamento_03.cpp

- Deriva da **Ordinamento_02.cpp**
- La funzione `Lettura` alloca dinamicamente il vettore, chiedendo all'utente quanti sono gli elementi

Programma: Ordinamento_03.cpp

```
void Lettura(int *&v, int &size)
{
    int i;

    cout << "ELementi? ";
    cin >> size;

    v = new int[size];
    if( v == NULL)
    {
        cout << "Memoria Esaurita";
        exit(1);
    }
```

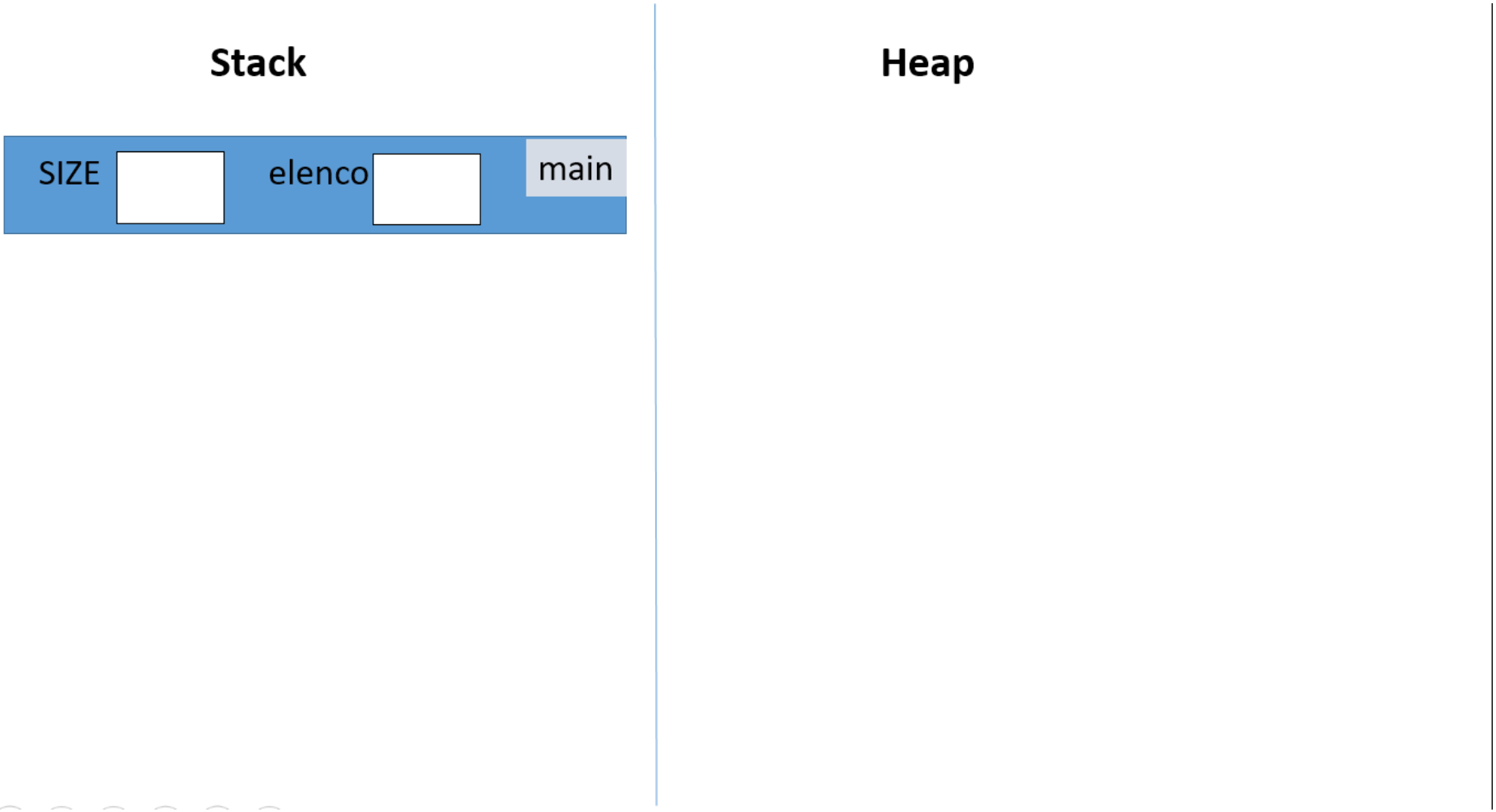
Programma: Ordinamento_03.cpp

```
    for(i=0; i < size; i++)
    {
        cout << "Inserire valore "
             << (i+1) << ": ";
        cin >> v[i];
    }
}
```

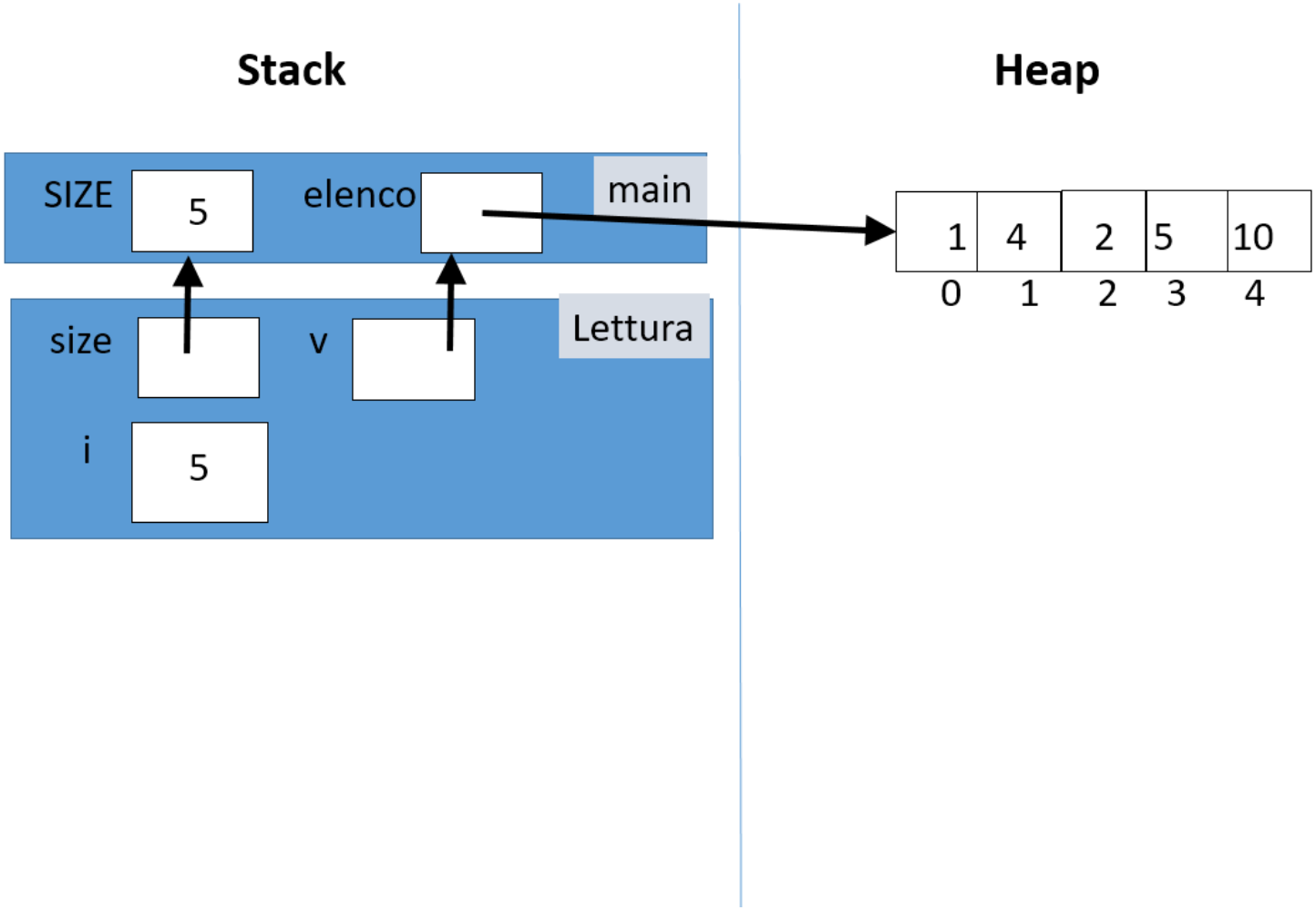
Programma: Ordinamento_03.cpp

```
int main()
{
    int SIZE;
    int *elenco;

    Lettura( elenco, SIZE);
    BubbleSort( elenco, SIZE );
    Stampa( elenco, SIZE );
    delete elenco;

   return 0;
}
```

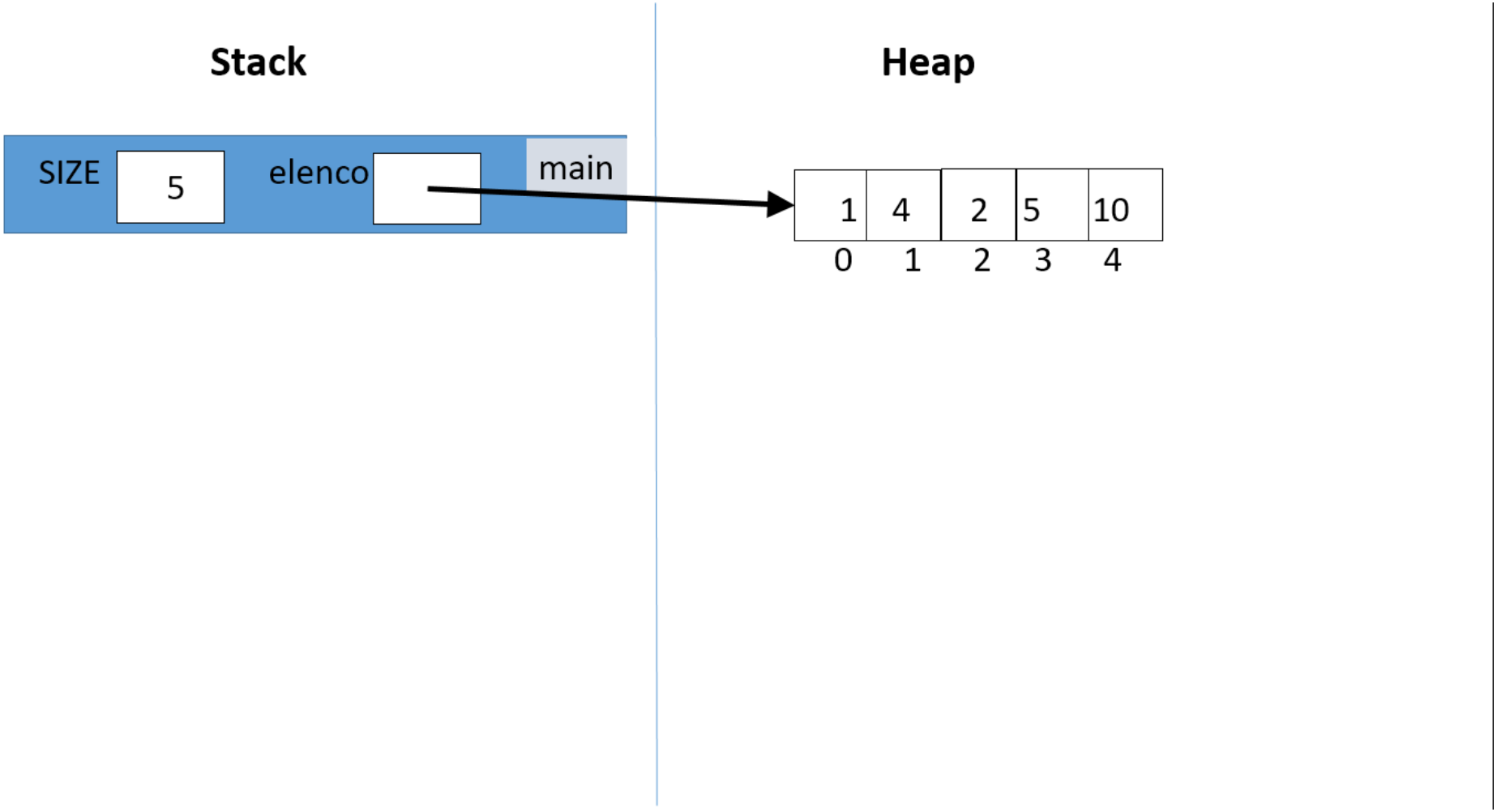
Memoria Centrale



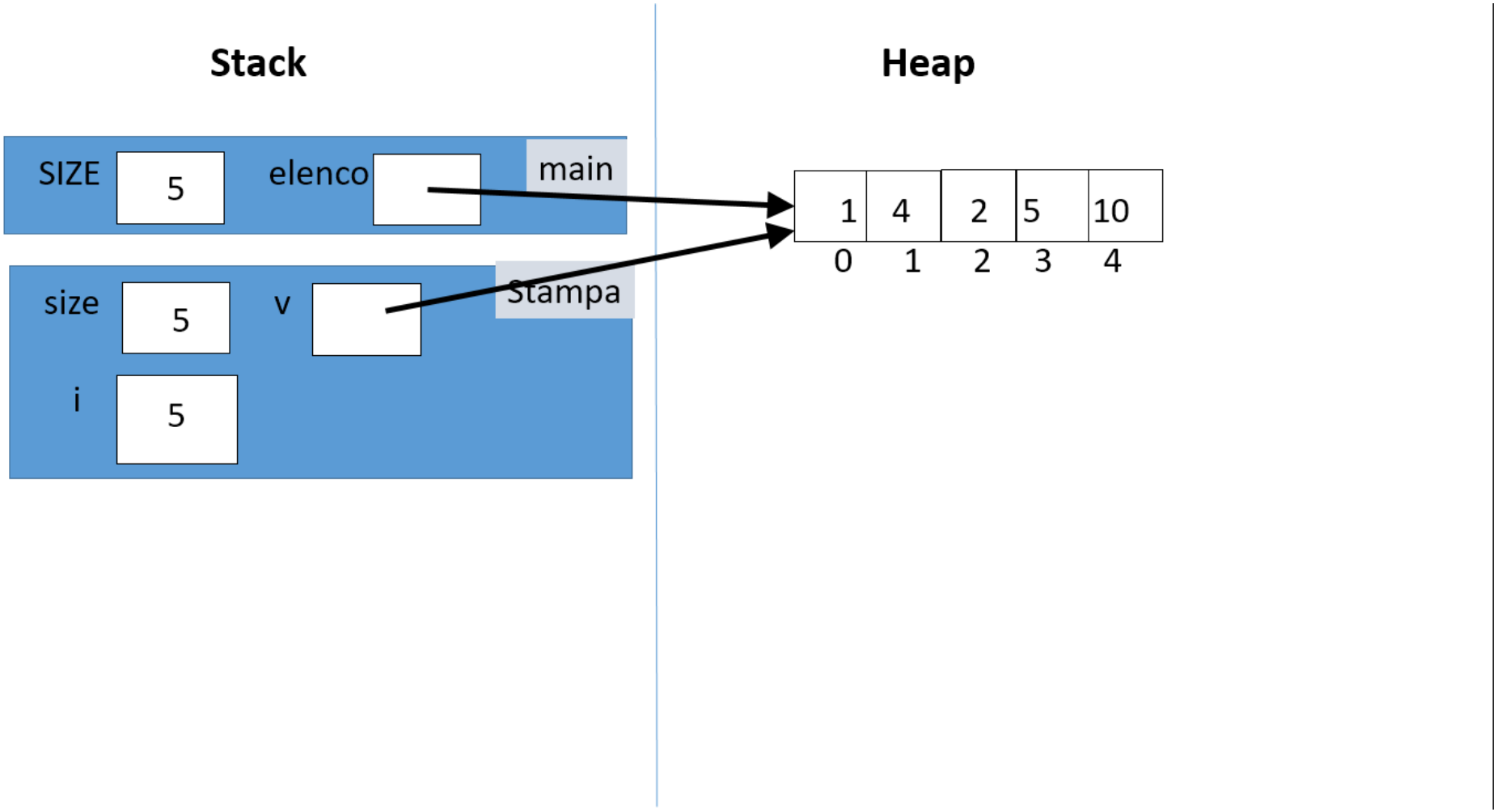
Memoria Centrale



Memoria Centrale



Memoria Centrale



Gestione File

- I file sono lo strumento con cui il programma può memorizzare in modo **PERSISTENTE** i dati elaborati
- Da un punto di vista logico, un file è una sorta di **Vettore**
- ma non sta in memoria centrale, quindi occorre usare una specifica libreria per accedervi: `fstream`

Gestione File

- Esistono due tipologie di gestione dei file:
  - ad **Accesso Sequenziale**
  - ad **Accesso Casuale**
- **Accesso Sequenziale**
  Si parte dall'inizio e si procede sempre in avanti, senza mai tornare indietro (stream, ruscello)
- **Accesso Casuale**
  Si salta all'interno del file, nella posizione desiderata, senza limitazioni
- Noi approfondiamo l'accesso sequenziale

Operazioni Sui File

- **Apertura**
  Richiede al sistema operativo di predisporsi per gestire il file, riservando opportuni spazi di memoria.
- **Uso**
  Il file viene usato (lettura o scrittura)
- **Chiusura del File**
  Si dice al sistema operativo che il file non deve essere più usato, le ultime modifiche vengono rese persistenti

Gestione a Blocchi

- Lo scambio di dati con il disco (o meglio, con il controller del disco) avviene a blocchi, non Byte per Byte
- Pertanto, il sistema operativo predispone un **Buffer**, uno spazio di memoria nel quale memorizzare i contenuti temporaneamente
- In lettura, un blocco viene caricato nel buffer e si legge da lì; quando il buffer è finito, si carica un altro blocco nel buffer
- In scrittura, i Byte vengono scritti nel buffer, quando questo si riempie, viene salvato su disco

Apertura del File ini Lettura

- Definire la variabile di tipo `ifstream`
  `ifstream fin;`
- Invocare la funzione `open` sulla variabile `fin`,
  `fin.open("dati.dat");`
- Controllare l'esito del tentativo di apertura, con la funzione `fail()` (Booleana)
  `if(fin.fail() ) ...`

Apertura del File ini Scrittura

- Definire la variabile di tipo `ofstream`
  `ofstream fout;`
- Invocare la funzione `open` sulla variabile `fout`,
  `fout.open("dati.dat", ios::out);`
  crea un nuovo file (o ne azzera uno esistente)
  `fout.open("dati.dat", ios::app);`
  accoda al file pre-esistente (o lo crea, se non esiste)
- Controllare l'esito del tentativo di apertura, con la funzione `fail()` (Booleana)
  `if(fout.fail() ) ...`

Chiusura del File

- Per qualsiasi tipo di file, invocare `close()`
- ```
  fin.close();
  fout.close();
  ```
  ```

Scrittura su File

- Come con `cout` (che è un file sempre aperto)
- `fout << ...;`
- Per forzare il trasferimento del buffer su disco `fout.flush();`

Esempio: Programma Scrittura_01.cpp

- Chiediamo all'utente un nome di file da creare
- Chiediamo all'utente quanti valori interi vuole salvare su file
- Leggiamo i valori e li scriviamo sul file

Programma: Scrittura_01.cpp

```
int main()
{
    int size, v, i;
    char nomefile[1000];
    ofstream fout;

    cout << "Nome del file: ";
    cin.getline(nomefile, 10000);
    cout << "Quanti Valori? ";
    cin >> size;

    fout.open(nomefile, ios::out);
    if(fout.fail())
    {
        cout << "Errore Apertura File";
        exit(1);
    }
```

Programma: Scrittura_01.cpp

```
    fout << size << endl;

    for(i=0; i < size; i++)
    {
        cout << "Valore " << (i+1) << ": ";
        cin >> v;
        fout << v << endl;
    }

    fout.close();

    return 0;
}
```

Lettura

- Si può usare l'operatore >>, come con `cin` (anche questo è un file sempre aperto).
- Esempio: Programma **Lettura_01.cpp** Leggiamo il file creato dal programma Scrittura_01.cpp

Programma: Lettura_01.cpp

```
int main()
{
    int size, v, i;
    char nomefile[1000];
    ifstream fin;

    cout << "Nome del file: ";
    cin.getline(nomefile, 10000);

    fin.open(nomefile);
    if(fin.fail())
    {
        cout << "Errore Apertura File";
        exit(1);
    }
```

Programma: Lettura_01.cpp

```
    fin >> size;

    for(i=0; i < size; i++)
    {
        fin >> v;
        cout << v << endl;
    }

    fin.close();

    return 0;
}
```

Lettura

- Per sapere se si è alla fine del file, si usa `fin.eof()` Booleana
- `while(!fin.eof()) ...`
- Vedi programma **Lettura_02.cpp**

Programma: Lettura_02.cpp

```
    while(!fin.eof() )
    {
        fin >> v;
        cout << v << endl;
    }

    fin.close();

    return 0;
}
```

Lettura

- Se sono presenti caratteri non numerici, si crea un problema: i caratteri non numerici non vengono letti e il programma si **INCHIODA** in un ciclo infinito
- Usare il programma **Scrittura_02.cpp** per scrivere una serie di punti e poi leggerlo con il programma **Lettura_02.cpp**

Programma: Scrittura_02.cpp

```
int main()
{
    int size, v, i;
    char nomefile[1000];
    ofstream fout;

    cout << "Nome del file: ";
    cin.getline(nomefile, 10000);
    cout << "Quanti Punti? ";
    cin >> size;

    fout.open(nomefile, ios::out);
    if(fout.fail())
    {
        cout << "Errore Apertura File";
        exit(1);
    }
```

Programma: Scrittura_02.cpp

```
    fout << size << endl;

    for(i=0; i < size; i++)
    {
        cout << "Punto " << (i+1) << "x: ";
        cin >> v;
        fout << v << ",";
        cout << "Punto " << (i+1) << "y: ";
        cin >> v;
        fout << v << endl;
    }

    fout.close();

    return 0;
}
```

Lettura

- Per risolvere il problema, occorre leggere **Solo Righe**, con `getline`, estraendo i valori dalla riga
- Vedi programma **Lettura_03.cpp**
- Il programma fa uso delle funzioni:
  `int atoi(char s[]);`
  converte il contenuto della stringa in numero intero
  `float atof(char s[]);`
  converte il contenuto della stringa in numero in virgola mobile
  `char * strncpy(char d[], char s[], int n);`
  copia solo i primi `n` caratteri (senza fine stringa)

Programma: Lettura_03.cpp

```
PUNTO estraiPunto(char r[])
{
    int virgola=-1;
    int i, l;
    char buffer[50];
    PUNTO p={0,0};

    for(i=0; r[i]!= '\0' && virgola < 0; i++)
    {
        if(r[i]==',')
            virgola=i;
    }
```

Programma: Lettura_03.cpp

```
        if(virgola >=0)
        {
            l = strlen(r);
            strncpy(buffer, r, virgola);
            buffer[virgola]='\0';
            p.x = atof(buffer);
            strncpy(buffer, r+virgola+1, l-virgola-1);
            buffer[l-virgola-1]='\0';
            p.y = atof(buffer);
        }
        return p;
}
```

Programma: Lettura_03.cpp

```
int main()
{
    int v, i, size;
    char nomefile[1000];
    ifstream fin;
    char riga[50];
    PUNTO p;

    cout << "Nome del file: ";
    cin.getline(nomefile, 10000);
```

Programma: Lettura_03.cpp

```
fin.open(nomefile);
if(fin.fail())
{
    cout << "Errore Apertura File";
    exit(1);
}

fin.getline(riga, 50);
size = atoi(riga);
```

Programma: Lettura_03.cpp

```
    for(i=0; i < size; i++)
    {
        fin.getline(riga, 50);
        p = estraiPunto(riga);
        stampa_Punto(p);
        cout << endl;
    }

    fin.close();

    return 0;
}
```

Variante

- Come fare per leggere un file che non indica, nella prima riga, quante righe devono essere lette?
- Il programma **Lettura_04.cpp** prova a leggere usando `eof()`
- Ma legge un punto in più. Perché?
- È colpa di Windows, che gestisce il 'fine riga' con due caratteri `CR` e `LF` (codici 13 e 10)
- `fin.getline` legge il `CR`, ma lascia il carattere `LF`, quindi `fin.eof()` restituisce 0, perchè c'è ancora un carattere

Programma: Lettura_04.cpp

```
int main()
{
    int v, i;
    char nomefile[1000];
    ifstream fin;

    cout << "Nome del file: ";
    cin.getline(nomefile, 10000);

    fin.open(nomefile);
    if(fin.fail())
    {
        cout << "Errore Apertura File";
        exit(1);
    }
```

Programma: Lettura_04.cpp

```
    while(!fin.eof() )
    {
        fin >> v;
        cout << v << endl;
    }

    fin.close();

    return 0;
}
```

Lettura

- Soluzione: contare quanti caratteri ci sono nel vettore `riga`
  perché se `getiline` non riesce a leggere, mete la stringa vuota
- ```
  fin.getline(riga, 50);
  letti = strlen(riga);
  while(!fin.fail() && letti>0)
  {
  ...
      fin.getline(riga, 50);
      letti = strlen(riga);
  }
  ```
  ```

Programma: Lettura_05.cpp

```
        fin.getline(riga, 50);
        letti = strlen(riga);
        while(!fin.fail() && letti>0)
        {
            p = estraiPunto(riga);
            stampa_Punto(p);
            cout << endl;
            fin.getline(riga, 50);
            letti = strlen(riga);
        }

        fin.close();

        return 0;
}
```

Lettura

- Lettura di singoli caratteri:
  `int fin.get(char & c)`
- Viene letto un carattere per volta, che viene memorizzato nel parametro ricevuto per reference
- Restituisce 1 se ha letto un carattere, 0 se non riesce a leggere
- Vedi progrmama **Copia.cpp**

Programma: Copia.cpp

```
int main(int argc, char*argv[])
{
    ifstream fin;
    ofstream fout;
    char c;

    if(argc != 3)
    {
        cout << "Argomenti Errati";
        return 1;
    }
```

Programma: Copia.cpp

```
fin.open(argv[1]);
if(fin.fail())
{
    cout << "Errore Apertura File" << argv[1];
    exit(1);
}
fout.open(argv[2], ios::out);
if(fout.fail())
{
    cout << "Errore Apertura File" << argv[2];
    exit(1);
}
```

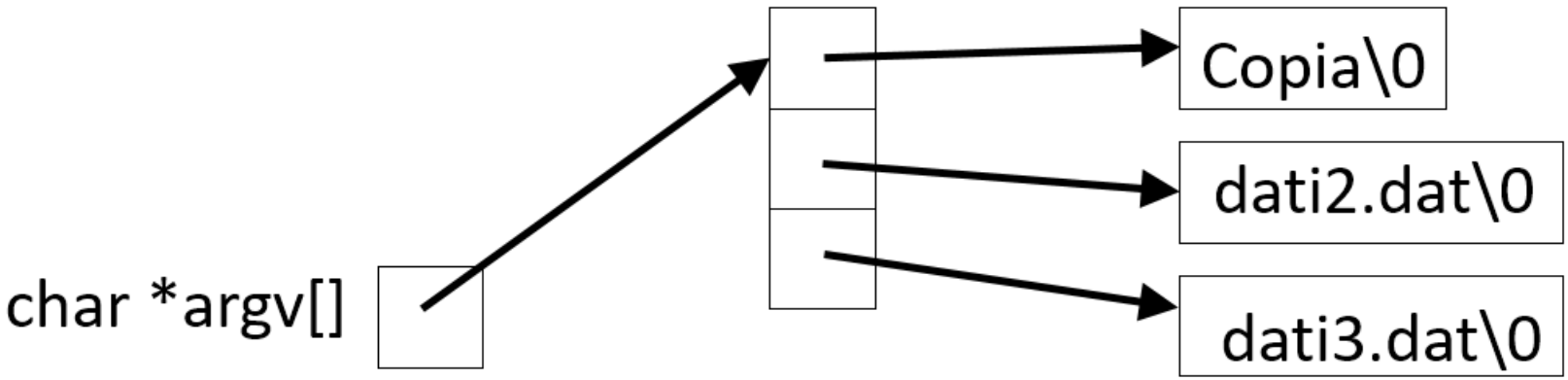
Programma: Copia.cpp

```
    while(fin.get(c))
    {
        fout << c;
    }

    fin.close();
    fout.close();

    return 0;
}
```

Argomenti sulla Linea di Comando

- Esempio:
  `copia dati2.dat dati3.dat`
- La riga viene decomposta in 3 parole;
- per riceverle, occorre aggiungere due parametri a `main`
- `int main(int argc, char *argv[])`
- `argc`: conta quanti sono gli argomenti
  `argv`: vettore di puntatori a carattere, tanti elementi quanti `argc`

Argomenti sulla Linea di Comando



Sfida

- Scrivere un programma che legge da tastiera un nome di file e un numero intero che indica quanti nominativi leggere;
- Leggere i nominativi nel numero indicato precedentemente;
- Salvarli sul file il cui nome è stato inserito all'inizio (un nominativo per riga) in ordine alfabetico.