**5° lezione pro II**

**Allocazione della memoria dinamica in C++**

* Le variabili globali occupano posizione fisse all’interno del segmento dati assegnato al programma dal sistema operativo
* Le variabili locali sono memorizzate nello STACK e vivono solo mentre è in esecuzione la funzione che le usa.
* Entrambe devono essere definite in fase compilazione. Non sempre è possibile sapere quando sarà necessario il programma
* Tramite le parole chiavi new e delate è possibile gestire la memoria in maniera dinamica. Le variabili dinamiche usano una zona della memoria chiamata HEAP.

L’operatore in

In genere dinamicamente una variabile di un certo tipo è assegna a essa un blocco di memoria della dimensione opportuna, inoltre, restituisce un contatore che contiene l’indirizzo del blocco di memoria allocato cioè della variabile, la quale sarà quindi accessibile referenziando il puntatore.

Ha la seguente sintassi:

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteImmagine che contiene testo, Carattere, bianco

Descrizione generata automaticamente

Immagine che contiene testo, ricevuta, Carattere, algebra

Descrizione generata automaticamente

È importante ricordare che l’ HEAP non è infinito infatti mancanza di spazio new restituisce NULL

**L'operatore delete**

L'operatore delete in C++ ha il seguente utilizzo:

* Libera la memoria allocata dinamicamente, rendendola disponibile per essere riallocata mediante successive chiamate all'operatore new.
* La sintassi dell'operatore delete è la seguente:
  + deleto puntatore; // per la deallocazione di singoli oggetti non array
  + delete [] puntatore; // per la deallocazione di array

Per esempio, lo spazio assegnato per le variabili dinamiche:

**int\* ad = new int;**

**char\* adc = new char[100];**

**si può liberare con le istruzioni:**

**delete ad;**

**delete [] adc;**

* L'operatore delete rende riutilizzabile la memoria puntata, ma non cancella il puntatore stesso, che può quindi essere riutilizzato per puntare a un'altra variabile successivamente allocata con new.

**Tipi di memoria in C++**

1. **Memoria automatica**: variabili definite all’interno di funzioni si dicono automatiche. Si allogano automaticamente nello Stack quando viene invocata la funzione in cui sono definite e si ingannano quando essa termina;

Sono ovviamente locali alla funzione in cui sono definite, sono locali dal blocco che ne contiene, cioè al più piccola sezione di codice rinchiusa tra parentesi graffa in cui sono definite

1. **Memoria statica**: l’allocazione statica avviene nel data Segment. Ci sono due modi per allocare staticamente una variabile
   1. Nel nulla al di fuori d di qualsiasi funzione inclusa anche la funzione main()
   2. Anteporre alla sua definizione della parola riservata static

3. **Memoria dinamica**: viene allocata o allocata a run-time nello heap dagli operatori new e delete. In assenza di un Garbage Collector(come Java e Visual Basic) la memoria rimane allocata fino a che viene affiancata dall’esecuzione di un’opportuna istruzione di delete.

**New e delete vs malloc() e free()**

Gli operatori new e delete in C++ hanno vantaggi significativi rispetto alle funzioni malloc() e free() utilizzate nel C. Ecco una comparazione dettagliata:

* **Efficienza**: Gli operatori new e delete sono generalmente più efficienti.
* **Tipo di Dato**: Allocano e deallocano memoria in funzione del tipo di dato da memorizzare, effettuando ogni volta i relativi controlli.
* **Operatore vs Funzione**: Sono operatori, non funzioni, quindi non necessitano di alcun header file.
* **No Casting**: Non richiedono il casting di tipo.
* **Costruttore e Distruttore**:
  + **new**alloca memoria dinamica ed esegue il costruttore della classe.
  + **malloc()** alloca solo memoria dinamica, senza eseguire alcun costruttore.
  + **delete**libera la memoria ed esegue il distruttore della classe.
  + **free()** libera solo la memoria, senza eseguire alcun distruttore.

**Gli stream o flussi**

**Concetti di Base**

* **Stream**: Un'astrazione che rappresenta un flusso di dati tra un programma produttore e un programma consumatore.
* **Standard Input**: Flusso di bytes che normalmente proviene dalla tastiera verso un programma in esecuzione sulla CPU.
* **Standard Output**: Flusso di dati che va dalla CPU verso un dispositivo di uscita.
* **Estrarre da un Flusso**: Ricezione di dati da un dispositivo di input.
* **Inserire in un Flusso**: Trasmissione di dati a un dispositivo di output.

**Header Files per Gestione degli Stream**

1. **<iostream>**: Contiene le classi istream, ostream e iostream per operazioni di I/O con i flussi standard e gli oggetti cout, cin, cerr e clog.
2. **<fstream>**: Contiene le classi ifstream, ofstream e fstream per operazioni di I/O su file.
3. **<sstream>**: Contiene le classi istringstream e ostringstream.
4. **<strstream>**: Contiene le classi istrstream, ostrstream e strstream per formattare dati con buffer di caratteri.

**Libreria <iostream>**

* **istream**: Permette di definire un flusso di input e contiene metodi per accettare dati sia formattati che non formattati. L'estrattore >> è sovraccaricato per tutti i tipi di dato fondamentali del C++ (e.g., int, float, string).

**File come Flussi**

* Un file è una sequenza di bytes in memoria di massa:
  + Se il file è aperto in scrittura, è un flusso di output.
  + Se il file è aperto in lettura, è un flusso di input.
* Classi principali per la gestione dei file:
  + ifstream: Per aprire un file in lettura.
  + ofstream: Per aprire un file in scrittura.
  + fstream: Per aprire un file sia in lettura che in scrittura.

**Esempi di Apertura di File**

* Apertura di un file in lettura:

ifstream fin("demo.txt");

* Apertura di un file in scrittura:

ofstream fout("demo.txt", ios::out); // Modalità output

ofstream fout("demo.txt", ios::app); // Modalità append

* Per default, un file ofstream si apre in modalità output:

ofstream fout("demo.txt");

Immagine che contiene testo, ricevuta, schermata, Carattere

Descrizione generata automaticamente

**Ridirezione degli Stream Standard**

* Ridirezionare lo stdout:

#include <stdio.h>

#include <iostream>

int main() {

freopen("myfile.txt", "w", stdout);

printf("Questa frase è ridirezionata su un file.\n");

std::cout << "Anche questa" << std::endl;

fclose(stdout);

return 0;

}

**I/O Binario**

* Per scrivere e leggere dati in formato binario si utilizzano le funzioni:
  + put()
  + get()
  + read()
  + write()

Esempio di utilizzo delle funzioni binarie:

#include <fstream>

int main() {

std::ofstream fout("data.bin", std::ios::binary);

int num = 12345;

fout.write(reinterpret\_cast<const char\*>(&num), sizeof(num));

fout.close();

std::ifstream fin("data.bin", std::ios::binary);

int num\_in;

fin.read(reinterpret\_cast<char\*>(&num\_in), sizeof(num\_in));

fin.close();

std::cout << "Numero letto: " << num\_in << std::endl;

return 0;

}