**12° lezione prog1**

Una **funzione** è una sequenza di istruzioni che viene identificata da un nome può ricevere dei dati in **ingresso o restituire un dato**. **Il prototipo di una funzione** è da intendersi come uno stampino su cui sono indicati i **parametri in ingresso e uscita.** **Esempio: int inserimento (int V[]){**

**return b;**

**}**

Nella funzione ci sono:

* **i parametri in ingresso**: sono quelli che passiamo alla funzione
* **I parametri in uscita**: sono quelli che restituisce la funzione nel caso della procedura restituirà 0 (void)

**Le dichiarazioni della funzione**

* **Tipo restituito** è il parametro di ritorno o parametro in uscita che puo restituire un valore in questo caso viene chiamata funzione se il tipo restituito è **void** la funzione non restituirà niente
* **nome della funzione** deve essere associato ad ogni funzione per spaziare una parola da un’altra si usa il trattino basso( \_ ).
* **Lista dei parametr**i è un elenco costituite da coppie (**tipo della variabile e l’identificatore della variabile stessa)**.

Esempio: **int inserimento (int V[]) {**

**return n;**

**}**

**I prototipi**

Sono intestazione delle procedure/funzioni e contengono sia i parametri in ingresso che quelli in uscita. **Esempio int pippo ( int a, int b**);

**Parametri attuali, formali e chiamata  di una funzione**

* + **La chiamata di una funzione**: è un istruzione che permette di interfacciare il programma con al funzione e solitamente si utilizza  nel main().
  + **Parametri attuali**: si hanno quando nel main si fa la “chiamata alla funzione o alla procedura” (cioè quando la si manda in esecuzione). Nella “chiamata” non possono esistere dichiarazioni (mai int, char..) ma solo passaggi di cose concrete come il vettore. **Non hanno il tipo.**
  + **Parametri formali**: si hanno  quando si dichiara una procedura (es.” procedura void carica(int v[], int n){}”).
  + E’ un elenco di coppie che è formato dal tipo della variabile da un identificatore  e permette  di interfacciare la funzione al programma chiamate indicando le variabili che dovrà elaborare.

**All'istruzione return** si può associare un argomento o un espressione nel caso in cui la funzione deve restituire un dato

Ci deve essere una corrispondenza tra i parametri attuali e quelli formali in termini di tipo non in termini di identificatori.

**Nel caso in cui si dovrebbe passare alla funzione un valore di un altro tipo viene eseguita una conversione implicita.** Nel caso di una funzione bisogna immagazzinare il dato restituito da questa in una variabile altrimenti questo verrebbe perso oppure va stampato tramite una funzione printf(). Si hanno dei casi in cui il parametro restituito non si utilizza quindi non è necessario la sua conservazione.

Nel caso in cui si debbano compilare ed eseguire più file in una volta si utilizza il comando gcc passandogli tutti i file sorgente ma questo potrebbe essere un procedimento lungo se si hanno tanti file da includere. Un'altra soluzione più efficiente sarebbe quella di utilizzare il compilatore gcc ma con -c in modo da generare un file oggetto che include molte definizioni ad esempio. successivamente utilizzare di nuovo il comando di cui sopra per generare un file oggetto che contiene anche la compilazione di altri file. Supponendo di avere un progetto con tanti file sorgente e di questi se ne modificano due non devo andare a compilare tutti i file sorgente ma solamente quelli che ho modificato(makefire). Un file object contiene codice macchina ma non codice eseguibile.

Le funzioni si scambiano i dati o tramite copia oppure tramite indirizzi.

**Lo stack è una struttura dati**, in cui si possono inserire oppure levare i dati quindi **push() e pop()** dalla cima o top dello stack. Si ha uno stack per l'invocazione delle funzioni o anche chiamato record di attivazione dello stack che è una porzione di memoria che si assegna alla funzione contestualmente all'esecuzione della prima istruzione di questa . Questa porzione di memoria viene de allocata alla fine della funzione. **Nello stack sono presenti le variabili locali e i parametri passati alla funzione**.

**Il return address** è l'indirizzo di ritorno della funzione si trova nel segmento data utilizzato per la memorizzazione delle istruzioni delle funzioni che **si utilizza per tornare ad eseguire l'istruzione successiva alla chiamata della funzione**.

**Immagine che contiene testo, schermata, Carattere, diagramma

Descrizione generata automaticamente**

**Variabili locali e globali**

* + **Le variabili globali**: vengono dichiarate fuori da ogni funzione (dopo la definizione delle librerie o header). Se si volesse dichiarare una variabile in un'altra trastation unit si dovrebbe utilizzare l’istruzione exstern specificando il nome ed il valore.
  + **Le variabili locali**: vengono dichiarate all’interno della singola  funzione procedura  e muoiono con la singola funzione.

**Passaggio di parametri alle funzioni**

Si hanno diversi modi per passare dei parametri alle funzione/procedure

-   **Per valore:** Il valore attuale del dato viene copiato sul record di attivazione dello stack .

Pertanto dato che la funzione opera su una copia di a, non può modificarne il valore

**Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente**

* + **Passaggio mediante indirizzo.** La funzione riceve l’indirizzo del dato (il puntatore), quindi può operare modifiche al dato della “funzione chiamante” mediante l’operatore di derefenziazione o indirezione.

Immagine che contiene testo, schermata, Carattere, ricevuta

Descrizione generata automaticamente

**Allocazione automatica dinamica e statica**

* **Allocazione automatica:** si dichiarano variabili locali alle funzioni si ha uno scope limitato al blocco di codice in cui è stata dichiarata e il ciclo di vita  del blocco allocato in memoria termina quando finisce l’  esecuzione  della funzione  **(STACK).**Immagine che contiene testo, Carattere, schermata, bianco

  Descrizione generata automaticamente
* **Allocazione dinamica:** si usa la funzione malloc() con la  libreria stdlib.h in qualunque punto del programma **(HEAP). I**l ciclo di vita del blocco di memoria termina con l'invocazione della funzione **free()** sul puntatoreImmagine che contiene testo, Carattere, schermata, bianco

  Descrizione generata automaticamente
* **Allocazione statica:** le variabili si dichiarano al di fuori di qualunque  blocco lo scope inizia e termina con il programma stesso e **la variabile globale(DATA)**

**Passaggio di array/vettori a funzioni(array con allocazione sia dinamica che automatica)**

 per passare alla funzione un’ array come parametro si hanno due modi:

1. **void caricamento(int \*v int n){}**
2. **void caricamento(int v[] int n){}**

L’array  è passato come parametro ad una  funzione sempre **mediante indirizzo** in particolare il **nome dell’ array  è un puntatore costante al primo elemento di questo**.

**Passaggio di array k dimensionali  (vale solo per allocazione automatica)**

si dichiararono in questo modo:

**Immagine che contiene testo, Carattere, bianco, algebra

Descrizione generata automaticamente**

**Nell’array k dimensionale vanno specificate  tutte le dimensioni dalla seconda in poi.**

**Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente**

**Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente**

**Se volessi utilizzare il VLA(Variable Lenght Array) nelle funzioni dovrei prima dare come parametro/parametri la dimensione e poi definire l’array. ES void insert\_elemets(int N, int V[N]);**