Funzioni come parametri

E’ possibile passare parametri alle funzioni/procedure:

- per copia (unico metodo supportato dal C)
- per «riferimento» (passando *copia* del puntatore alla variabile)

MA…

- Non è possibile passare a funzioni/procedure altre funzioni come parametro…
- Non è possibile che una funzione «restituisca» come risultato un’altra funzione/procedura

Funzioni come parametri

Esempio:

Realizzare una funzione somma(…) che, data una qualunque funzione f:R-->R, restituisca la somma per i primi 100 numeri x di f(x).

$$somma = \sum_{i=1}^{100} f(x)$$

Problema: per realizzare tale programma, dobbiamo conoscere in anticipo f(x)… ma è proprio necessario?

Puntatori a Funzione

Soluzione: potremmo usare un nuovo tipo detto puntatore a funzione e passare una avariabile di tale tipo come parametro alle funzioni/procedure…

PUNTATORE A FUNZIONE

<tipo_f> (* <nomeF>)(<lista_par_formali>);

dove:

- <nomeF> è l'identificatore del puntatore a funzione;
- <tipo_f> è il tipo di dato restituito dalla funzione puntata
- <lista_par_formali> è la lista dei parametri formali della funzione puntata

Puntatori a Funzione - Esempio

double (* **f**) (double par);

f è un puntatore a una funzione che riceve in ingresso un parametro di tipo double e restituisce un double

Attenzione: nel linguaggio C l'identificatore di una funzione è identificato come un *puntatore al suo codice*.

Nota: priorità di () rispetto a *….

Funzioni come parametri

Tipo1 f1(param1, ???, param2);

Tipo1 f1(param1, Tipo2 (* f2)(lista_parametri_di_f2), param2);

La funzione f1(…) riceve come parametro formale un puntatore a funzione, di nome f2

Nel corpo di f1(…) è possibile chiamare la funzione puntata da f2 tramite il *dereferencing*:

```
Tipo1 F1(param1, Tipo2 (*F2)(lista-par-formali-F2), param2) {
        tipo2 k;
        ...
        k = *F2(...); /*chiamata mediante dereferencing*/
}
```

Puntatori a Funzione

- Un parametro formale funzione è specificato indicando un nome (puntatore), la lista dei parametri formali ed il tipo di risultato.
- Il parametro attuale con cui si invoca f1 deve essere un identificatore di funzione con la **stessa descrizione dei parametri e lo stesso tipo di risultato di f2.**

Esempio

```
#include <math.h>
double fun (double x) {
        return x*x;
}

double somma( double (*f)(double par), int start, int end) {
        int val;
        double result = 0.0;
        for (val=start; val<end; i++)
                result = result + (*f)(val);
        return result;
}


int main() {
        double num1,num2;
        ...
        num1 = somma(fun, 1, 1000);
        num2 = somma(sin, 1, 1000);
        ...
}
```

Tipi di Puntatori a Funzione

- E' possibile dichiarare un *tipo funzione*, da utilizzare poi per la dichiarazione di variabili o come identificatore di tipo di un parametro formale, o risultato di un altra funzione.

Si definiscono, in pratica, come variabili di tipo puntatore a funzione:

```
typedef int fprot1 (int a, int b); /*tipo funzione */
typedef fprot1 * funptr;       /*tipo punt. a funzione*/
```

Tipi di Puntatori a Funzione

Ad esempio:

```
typedef double (* ptrfunzione) (double);

ptrfunzione v1ptr;
```

Variabili di questo tipo possono essere usate in assegnamenti:

```
v1ptr = &sin;
```

Nota: questo assegnamento non implica alcuna valutazione della funzione.

Funzioni come risultato di funzioni

Una funzione può restituire un puntatore a funzione.

Esempio:

```
typedef int fprot1 (int a, int b);

typedef fprot1 *funptr;

fprot1 *select1 (int a) {...}
```

La funzione select1 ritorna un puntatore a funzione del tipo a due parametri interi e un intero di ritorno

In alternativa:

```
funptr  select2 (int a) {...}
```

oppure:

```
int (* select3(int a)) (int a, int b) {...}
```

Esempio

```
#include <stdio.h>

typedef int fprot1 (int a, int b);
typedef fprot1 *funptr;

int selecta (int a, int b)
   { return a + b; }

int selectb (int a, int b)
   { return a - b; }

int selectc (int a, int b)
   { return a * b; }

...
```

Esempio

```
...

funptr select2 (int a)
{ switch (a)
   { case 1: return selecta; break;
         case 2: return selectb; break;
         case 3: return selectc; break;
         default: return selecta;
   }
}

main()
{  int x, y, scelta;
   funptr F;
   printf("Quale funzione (1,2,3)? ");
   scanf("%d", &scelta);
   printf("Valori? ");
   scanf("%d%d", &x, &y);
   F=select2(scelta);
   printf("Risultato:%d\n",(*F)(x,y));
}
```