

I puntatori

Una variabile di tipo **puntatore** contiene l'indirizzo di memoria di un'altra variabile:

int *a → puntatore contenente l'indirizzo di una variabile intera

char *s → puntatore di tipo carattere

struct data *x → puntatore per una struttura ^(dato) definita dall'utente

L'operatore **&** permette di conoscere l'indirizzo di memoria dove è memorizzata una variabile

int num;

int *p;

p = # → p punta alla memoria occupata da num

→ indirizzo di memoria dove è memorizzata la variabile intera "num"

Per accedere al valore della memoria puntata da un puntatore si usa l'operazione *****:

int a, b;

int *p;

a = 5;

p = &a;

b = *p; → b = 5

Esempio:

...

int a = 5; } Valore della variabile a (`printf("%d", a);`) → 5

int *p = &a; } Indirizzo della variabile a (`printf("%p", &a);`) → 0042043C

... Valore del puntatore ad a (`printf("%p", p);`) → 0042043C

Indirizzo del puntatore ad a (`printf("%p", &p);`) → 00420440

Valore della variabile puntata (`printf("%d", *p);`) → 5

Passaggio di parametri per indirizzo:

prototipo: **void modifica (int &a, int &b);**

chiamata: **int x = 5, y = 6**

modifica(x, y);

Negli array

int v[10];

int *p;

p = v; (oppure **p = &v[0]** → 'v' è per definizione **&v[0]**)

Nelle funzioni

void modifica(int *a); equivalente a **void modifica(int a[])**

a[i] = x equivale a ***(&a + i) = x**

Aritmetica dei puntatori:

int a[10];

int *p = a; (→ ***p = a[0]**)

p++; (→ ***p = a[1]**)

↳ sposta il puntatore in avanti di un numero di posizioni pari alla dimensione del tipo a cui punta (int → 4 byte)

Puntatore a struttura

struct Data {

int giorno;

char [10] mese;

int anno;

}

struct Data x; → è necessario allocare i dati per far sì che essi occupino memoria

struct Data *p;

p = &x;