UNIVERSITÀ DEGLI STUDI DI BERGAMO

Informatica
Modulo di Programmazione

Informatica
Modulo di Programmazione
Pointers

mauro.pelucchi@gmail.com
Mauro Pelucchi

2023/2024

Agenda

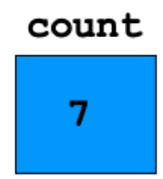
- Pointers
- Structs

Pointers

- Pointers
  - Powerful, but difficult to master
  - Simulate pass-by-reference
  - Close relationship with arrays and strings
- Can declare pointers to any data type
- Pointer initialization
  - Initialized to **`0`**, **`NULL`**, or address
    - **`0`** or **`NULL`** points to nothing

Pointer Variable Declarations and Initialization

- Pointer variables
  - Contain memory addresses as values
  - Normally, variable contains specific value (direct reference)
  - Pointers contain address of variable that has specific value (indirect reference)



- Indirection
  - Referencing value through pointer
- Pointer declarations
  - `*` indicates variable is pointer

```
int *myPtr;
```

    declares pointer to `int`, pointer of type `int *`
  - Multiple pointers require multiple asterisks

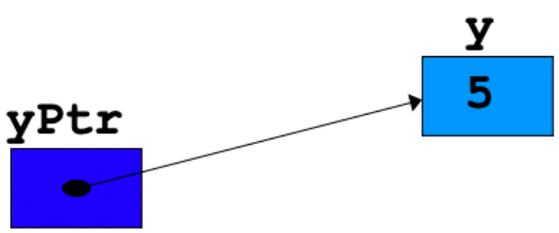
```
int *myPtr1, *myPtr2;
```

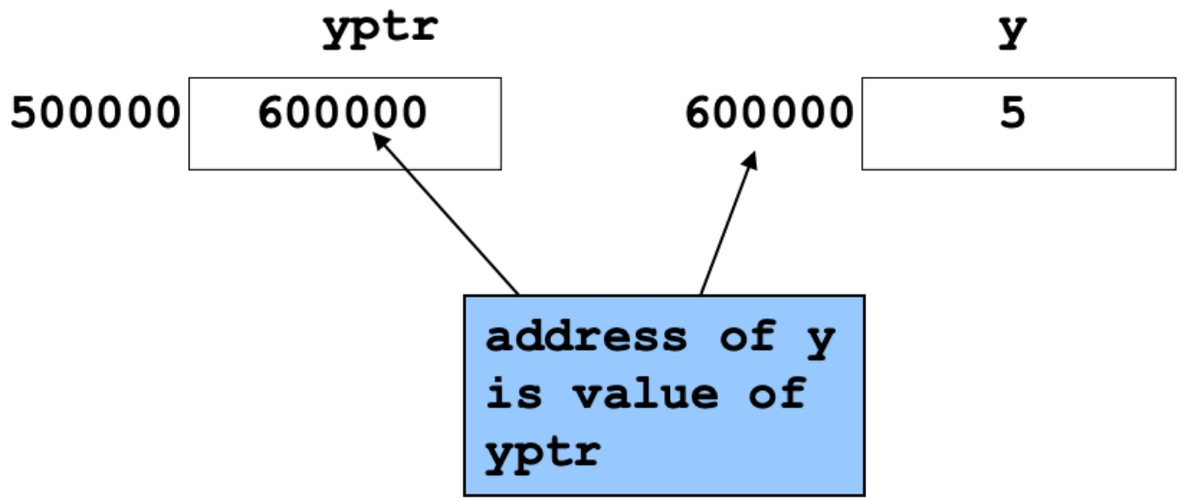
Pointers Operators

- **&** (address operator)
  - Returns memory address of its operand
  - Example

```
int y = 5;
int *yPtr;
yPtr = &y;     // yPtr gets address of y
```

  - **yPtr** "points to" **y**





Pointers Operators

- **`*`** (indirection/dereferencing operator)
  - Returns synonym for object its pointer operand points to
  - **`*yPtr`** returns **`y`** (because **`yPtr`** points to **`y`**).
  - dereferenced pointer is lvalue

```
*yptr = 9;          // assigns 9 to y
```

- **`*`** and **`&`** are inverses of each other

Using const with Pointers

- **`const`** qualifier
  - Value of variable should not be modified
  - **`const`** used when function does not need to change a variable
- Principle of least privilege
  - Award function enough access to accomplish task, but no more
- Four ways to pass pointer to function
  - Nonconstant pointer to nonconstant data
    - Highest amount of access
  - Nonconstant pointer to constant data
  - Constant pointer to nonconstant data
  - Constant pointer to constant data
    - Least amount of access

Using const with Pointers

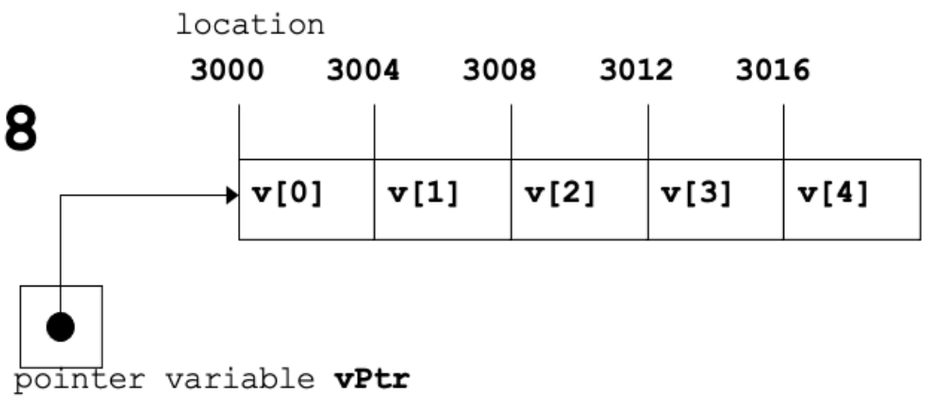
- **`const`** pointers
  - Always point to same memory location
  - Default for array name
  - Must be initialized when declared

Pointer Expressions and Pointer Arithmetic

- Pointer arithmetic
  - Increment/decrement pointer (**++** or **--**)
  - Add/subtract an integer to/from a pointer(**+** or **+=** , **-** or **-=**)
  - Pointers may be subtracted from each other
  - Pointer arithmetic meaningless unless performed on pointer to array
- 5 element **int** array on a machine using 4byte **int**s
  - **vPtr** points to first element **v[0]**, which is at location 3000

    vPtr = 3000
  - **vPtr += 2**; sets **vPtr** to **3008**

    vPtr points to **v[2]**



Pointer Expressions and Pointer Arithmetic

- Subtracting pointers
  - Returns number of elements between two addresses

```
vPtr2 = v[ 2 ];
vPtr = v[ 0 ];
vPtr2 - vPtr == 2
```

- Pointer assignment
  - Pointer can be assigned to another pointer if both of same type
  - If not same type, cast operator must be used
  - Exception: pointer to **`void`** (type **`void *`**)
    - Generic pointer, represents any type
    - No casting needed to convert pointer to **`void`** pointer
    - **`void`** pointers cannot be dereferenced

Pointer Expressions and Pointer Arithmetic

- Pointer comparison
  - Use equality and relational operators
  - Comparisons meaningless unless pointers point to members of same array
  - Compare addresses stored in pointers
  - Example: could show that one pointer points to higher numbered element of array than other pointer
  - Common use to determine whether pointer is 0 (does not point to anything)

Arrays & Pointers

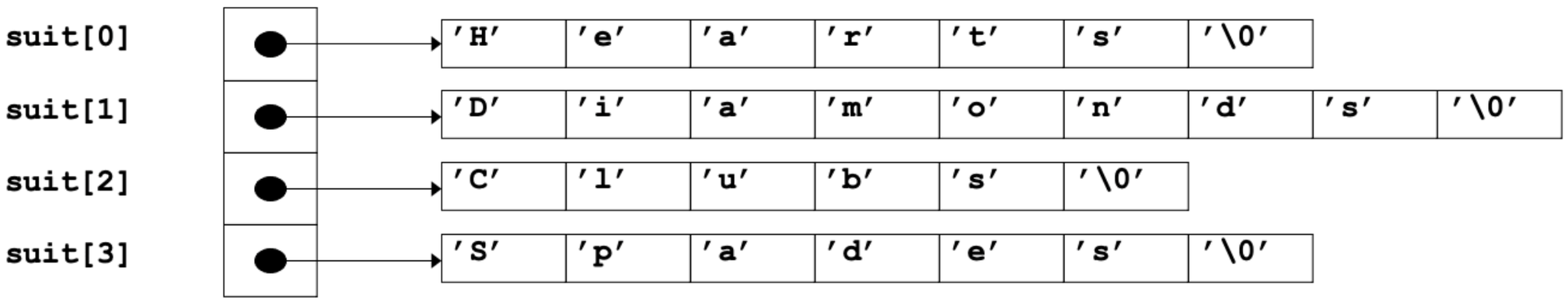
- Arrays and pointers closely related
  - Array name like constant pointer
  - Pointers can do array subscripting operations
- Accessing array elements with pointers
  - Element **`b[ n ]`** can be accessed by **`*( bPtr + n )`**
    - Called pointer/offset notation
  - Addresses
    - **`&b[ 3 ]`** same as **`bPtr + 3`**
  - Array name can be treated as pointer
    - **`b[ 3 ]`** same as **`*( b + 3 )`**
  - Pointers can be subscripted (pointer/subscript notation)
    - **`bPtr[ 3 ]`** same as **`b[ 3 ]`**

Arrays of Pointers

- Arrays can contain pointers
  - Commonly used to store array of strings

```
char *suit[ 4 ] = {"Hearts", "Diamonds",
                   "Clubs", "Spades" };
```

  - Each element of **suit** points to **`char *`** (a string)
  - Array does not store strings, only pointers to strings



  - **`suit`** array has fixed size, but strings can be of any size

Function Pointers

- Calling functions using pointers
  - Assume parameter:
    - **`bool ( *compare ) ( int, int )`**
  - Execute function with either
    - **`( *compare ) ( int1, int2 )`**
      - Dereference pointer to function to execute

OR

- **`compare( int1, int2 )`**
  - Could be confusing
    - User may think **`compare`** name of actual function in program

Structure Definitions

- Structures
  - Aggregate data types built using elements of other types

```
struct Time {
        int hour;
        int minute;
        int second;
};
```

Structure tag

Structure members

- Structure member naming
  - In same **`struct`**: must have unique names
  - In different **`struct`**s: can share name
- **`struct`** definition must end with semicolon

Structure

- Member access operators
  - Dot operator (**.**) for structure and class members
  - Arrow operator (**->**) for structure and class members via pointer to object
  - Print member **hour** of **timeObject**:

```
cout << timeObject.hour;
```

OR

```
timePtr = &timeObject;
cout << timePtr->hour;
```

  - **timePtr->hour** same as **(*timePtr).hour**
    - Parentheses required
      - ***** lower precedence than **.**

Esercizio 1

La seguente dichiarazione definisce il catalogo prezzi nel sistema informativo di un piccolo supermercato, su una piattaforma in cui

```
sizeof(long)==8,  sizeof(float)==sizeof(int)==sizeof(void *)==4

typedef char Descrizione[40];
typedef struct { unsigned long barcode;
                 Descrizione descr;
                 float prezzo;
                 int noPunti;
                 char * descrEstesa;
               } Articolo;
typedef struct {
                 Articolo list[1000];
                 int numArticoli;
               } CatalogoGenerale;

CatalogoGenerale cat;
```

Esercizio 1

noPunti è un booleano che identifica gli articoli su cui per legge non si possono effettuare promozioni o fidelizzazioni; **numArticoli** indica quanti dei 1000 elementi di list sono effettivamente utilizzati.

(a) Si calcoli quanto vale **sizeof(cat)**
(b) Sapendo che **cat** è allocato a partire dall' indirizzo 2348, si calcoli l'indirizzo della cella contenente la variabile **cat.list[20].noPunti**
(c) Si implementi una funzione C++ …**stampa(...)** che riceve come parametro un Articolo (passato per indirizzo) e stampa su stdout una riga così come deve apparire sullo scontrino.
(d) Si consideri che:
(1.) i cataloghi contengono gli articoli in ordine di codice a barre crescente, e
(2.) non possono esservi due articoli con lo stesso codice a barre.

Si implementi una funzione C **…trova(...)**
che riceve come parametri un catalogo di articoli (passato per indirizzo) e un codice a barre, e restituisce al chiamante l'indirizzo dell'articolo identificato dal codice, o NULL se il codice non corrisponde a nessun articolo in catalogo.

Esercizio 1

(e) Si implementi una funzione **...scan(...)** che legge da stdin una sequenza di interi positivi (di lunghezza ignota e a priori illimitata), terminata dal valore 0, che rappresentano i codici a barre nell'ordine in cui sono letti dallo scanner laser di una delle casse del supermercato.
La funzione riceve come parametro il catalogo del supermercato (passato per indirizzo), e:

- Restituisce al chiamante l' importo totale da pagare.
- Stampa su stdout lo scontrino (attenzione: non si perda tempo a organizzare intestazioni elaborate!).
- Stampa su stdout i punti fedeltà conseguiti dal cliente (si ha diritto a un punto fedeltà per ogni euro di spesa oltre i primi 5euro, senza considerare i prodotti "non promozionabili", come ad esempio giornali e medicinali, per i quali l'attributo noPunti ha valore 1).

f) Si spieghi brevemente perché è particolarmente opportuno che trova() e scan() ricevano il catalogo per indirizzo.

...

Esercizio 2

Dichiarare un tipo per rappresentare i punti in un Piano Cartesiano.

a) Realizzare un sottoprogramma per l'acquisizione da stdin di un punto del Piano Cartesiano, esibendo due versioni del sottoprogramma: procedurale e funzionale:

```
void procedureGetPoint(Point *p)
  Point functionGetPoint(void)
```

b) Realizzare un sotto-programma

```
Point meanPoint(Point p1, Point p2)
```

che dati due punti restituisce il punto medio del segmento che li congiunge.

c) Realizzare un sotto-programma

```
double distance(Point p1, Point p2)
```

che dati due punti restituisce la distanza tra i due.

Esercizio 2

Un triangolo può essere rappresentato con un vettore di tre punti:

```
typedef Point Triangle[3]
```

d) Realizzare un sotto-programma per l'acquisizione delle coordinate dei vertici di un triangolo nelle due versioni:

```
void GetTriangleWithFunction(Triangle t)
void GetTriangleWithProcedure(Point *t)
```

e) Scrivere un sotto-programma che verifichi se due triangoli sono uguali.

```
int testTriangles(Triangle t1, Triangle t2)
```

Esercizio 4

Scrivere una funzione **conc_pars(…)** che riceva come parametri due stringhe, e restituisca come risultato una **nuova stringa, ottenuta dalla concatenazione delle stringhe ricevute**, facendo in modo che la nuova stringa occupi la quantità minima di memoria.
La funzione non deve modificare le stringhe ricevute, ma restituisce un puntatore alla nuova stringa, che deve essere allocata dinamicamente (o non potrebbe essere utilmente restituita dalla funzione).
Esempio: s1 = "Prima" s2 = "Seconda" -> restituisce = "PrimaSeconda"

Si scriva poi una procedura void **strmul(…)** che, data una stringa str (che dev'essere stata allocata dinamicamente) e un numero intero n, modifichi la stringa str, riallocandola e assegnandole il risultato della concatenazione di str con se stessa per n volte.
Si noti che str deve essere stata allocata dinamicamente (o la funzione non potrebbe deallocarla e sostituirla con una più lunga!)
str deve essere opportunamente riallocata in modo da contenere esattamente il risultato della "moltiplicazione" (dimensione minima)
per essere modificata dalla funzione, la stringa str (che nell'ambiente del programma chiamante è un puntatore a carattere) deve essere passata per reference oppure per indirizzo.

Esempio: s = "abC" -> moltiplicazione per 3 -> s = "abCabCabC"

Suggerimento:
si possono usare le funzioni strcat() e/o conc_pars() ?

Esercizio 5

Si consideri un vettore che rappresenta un elenco di N stringhe definito come segue.

```
#define N 4
typedef char parola[20+1];
typedef parola elenco[N];
```

Data una variabile: elenco x;
acquisire dallo standard input N stringhe e riportarle poi a video in ordine alfabetico.

Memento: la funzione int strcmp(char s1[], char s2[]) confronta alfabeticamente s1 e s2.

Esercizio 6

Ci ispiriamo alla funzione **strcmp(...).** Si codifichi una funzione **datecmp(...)** che stabilisce se due date, passate come parametri, sono uguali (nel qual caso restituisce 0), oppure in ordine cronologico (e allora restituisce un numero negativo), oppure in ordine cronologico inverso (restituendo un numero positivo).

Si codifichi inoltre una funzione che stampa una data nel formato **aaaa/mm/gg**
Si scriva anche un piccolo programma (**main**) per verificare che le funzioni siano state realizzate correttamente.

Si utilizzino le seguenti definizioni:

Esercizio 6

```
struct Data { int giorno;
              int mese;
              int anno;
};

int datecmp( Data d1, Data d2 ) {
        /* ... */
}

void stampa( Data d ) {
        /* ... */
}

void stampaConLettere( Data d ) {
        /* stampa le date in formato
              15 Mar 44 a.C.
              13 Ott 2013 d.C.
      */
}
```

Esercizio 7

Si definisce il tipo di dato **NumeroCodificato** come segue:

```
struct NumeroCodificato { int base;
                          char cifre[20];
};
```

nel quale la **base** è un intero compreso tra 2 e 10 e **cifre** è una stringa (terminata da '\0') contenente i caratteri corrispondenti alle cifre ('0', '1', …) della codifica del numero in base **base**.
Si definiscano, codifichino e testino le funzioni:

```
int convertiInt(NumeroCodificato n);
```

che calcola il valore del numero n e lo restituisce espresso come intero

```
void stampaDec(NumeroCodificato n);
```

che stampa a video il numero n in notazione decimale

```
NumeroCodificato codifica(int n, int b);
```

che codifica l'intero n in un NumeroCodificato di base b e lo restituisce al chiamante

```
NumeroCodificato converti(NumeroCodificato n,int b2);
```

che converte il numero n in un altro, codificato in base b2, e lo restituisce al chiamante