Passaggio di parametri per valore e per riferimento

Liceo G.B. Brocchi
Classi seconde Scientifico – opzione scienze applicate
Bassano del Grappa, Gennaio 2023
Prof. Giovanni Mazzocchin

Passaggio di parametri per valore

```
void f (int x_arg) {
        x_arg++;
        cout << "value of actual parameter is: " << x_arg << endl;
}

int main (int argc, char* argv[]) {
        int x_main = 6;
        f(x_main);
        cout << "value of main's variable is: " << x_main << endl;

        return 0;
}
```

```
value of actual parameter is: 7
value of main's variable is: 6
```

Perché il valore di x_main non è cambiato?

Passaggio di parametri per valore

```
void f (int x_arg) {
        x_arg++;
        cout << "value of actual parameter is: " << x_arg << endl;
}

int main (int argc, char* argv[]) {
        int x_main = 6;
        f(x_main);
        cout << "value of main's variable is: " << x_main << endl;

        return 0;
}
```

Activation record di f — **x_arg: copia del valore di x_main**

Activation record di main — **x_main: 6**

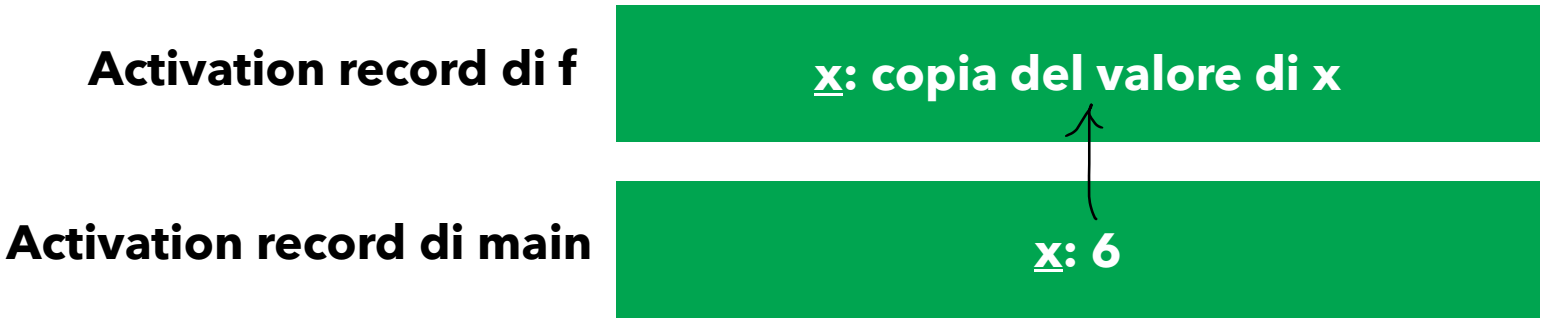
Passaggio di parametri per valore

```
void f (int x) {
        x++;
        cout << "value of actual parameter is: " << x << endl;
}

int main (int argc, char* argv[]) {
        int x = 6;
        f(x);
        cout << "value of main's variable is: " << x << endl;

        return 0;
}
```

x di f e x di main sono 2 variabili diverse. x di f prende solo il valore di x di main



Passaggio di parametri per valore

```
void f (int x) {
        x++;
        cout << "address of f's x is: " << &x << endl;
        cout << "value of actual parameter is: " << x << endl;
}

int main (int argc, char* argv[]) {
        int x = 6;
        cout << "address of main's x is: " << &x << endl;
        f(x);
        cout << "value of x is: " << x << endl;
}
```

x di f e x di main sono 2 variabili diverse. x di f prende solo il valore di x di main

```
address of main's x is: 0093FF24
address of f's x is:    0093FF20
```

indirizzi diversi, posizioni di memoria diverse, quindi variabili diverse

Passaggio di parametri per valore

```
void f (int* xptr) {
        *xptr = *xptr + 1;
}

int main (int argc, char* argv[]) {
        int x = 9;
        f(&x);
        cout << "value of x is: " << x << endl;
        return 0;
}
```

Passaggio di parametri per valore

```
void f (int* xptr) {
        *xptr = *xptr + 1;
        cout << "value of xptr parameter in f function is: " << xptr << endl;
}

int main (int argc, char* argv[]) {
        int x = 9;
        cout << "initial value of x in the main function is: " << x << endl;
        cout << "address of x in the main function is: " << &x << endl;
        f(&x);
        cout << "value of x is: " << x << endl;

        return 0;
}
```

```
initial value of x in the main function is: 9
address of x in the main function is:     008FFE20
value of xptr parameter in f function is: 008FFE20
value of x is: 10
```

Riflettere sul fatto che viene stampata la stessa cosa. Spiegare e rispiegare il perché.

Passaggio di parametri per valore

```
void f (int* xptr, int inc) {
        *xptr = *xptr + inc;
}

int main (int argc, char* argv[]) {
        //call f in the right way
}
```

Passaggio di parametri per valore

```
void f (int* xptr, int inc) {
        *xptr = *xptr + inc;
}

int main (int argc, char* argv[]) {
        //call f in the right way
}
```

Passaggio di array a funzioni

```
void u (char s[], int index_1, int index_2) {
        s[index_1] = toupper(s[index_1]);
        s[index_2] = toupper(s[index_2]);
}

int main() {
        char str[14] = {'s', 'i', 'l', 'i', 'c', 'o', 'n', 'v', 'a', 'l', 'l', 'e', 'y', '\0'}
        cout << "str's content before calling u is: " << str << endl;
        u(str, 0, 7);
        cout << "str's content after calling u is: " << str << endl;
}
```

- la funzione **toupper** restituisce l'uppercase di un carattere ('a' → 'A')

Passaggio di array a funzioni

```
void u (char s[], int index_1, int index_2) {
        s[index_1] = toupper(s[index_1]);
        s[index_2] = toupper(s[index_2]);
}

int main() {
        char str[14] = {'s', 'i', 'l', 'i', 'c', 'o', 'n', 'v', 'a', 'l', 'l', 'e', 'y', '\0'};
        cout << "str's content before calling u is: " << str << endl;
        u(str, 0, 7);
        cout << "str's content after calling u is: " << str << endl;
}
```

```
str's content before calling u is: siliconvalley
str's content after calling u is: SiliconValley
```

Perché **cout << str** *stampa il contenuto dell'array? Con gli array di interi funzionava così?*

Passaggio di array a funzioni

```
void u (char* s, int index_1, int index_2) {
        s[index_1] = toupper(s[index_1]);
        s[index_2] = toupper(s[index_2]);
}
```

È uguale alla funzione di prima. Qui abbiamo dichiarato il parametro formale **s** come puntatore a carattere. Possiamo passare ad **u** un array di caratteri? Sì, perché un array è un puntatore.

Passaggio per riferimento

- Finora abbiamo utilizzato i **puntatori** per fare **side-effect** sui parametri di una funzione, ossia per modificarli e rendere la modifica permanente anche dopo che l'uscita dal blocco della funzione
- Sappiamo però che i puntatori sono scomodi, hanno una sintassi che abbiamo definito *antipatica*… poi ogni tanto salta fuori quel maledetto **null pointer** che causa disastri inimmaginabili…

- Abbiamo visto i riferimenti. Usiamoli per fare side-effect sui parametri!

Passaggio per riferimento

```
void multiplicator(int& n, int m) {
        n = n * m;
}

int main() {
    int start = 1;

    for (int i = 0; i < 20; i++) {
        cout << start << endl;
        multiplicator(start, 2);
    }
}
```

uguali!

```
address of start in main function is: 0133F968
address of n in multiplicator function is: 0133F968
```