Parameters: Pass-by-reference
Pass-by-value

Oct 17 '24
week 6

Recall:

parameters

```
void fnc(int a, float b);  // a prototype

int main()
{
    .
    .
    .
    fnc( partNum, qualMetric );
    .
    .
    .
}
```

int

float

call

```
void fnc(int b, float a)  // a stub defn
{

    return;
}
```

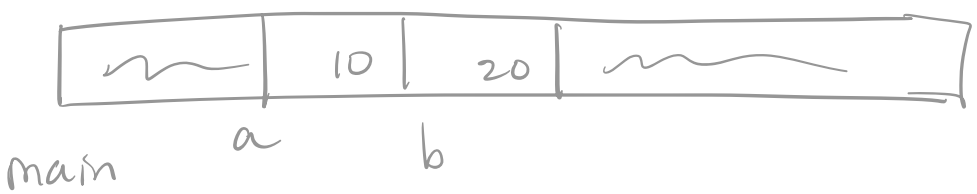
Aside: can have different names in defn than in prototype.

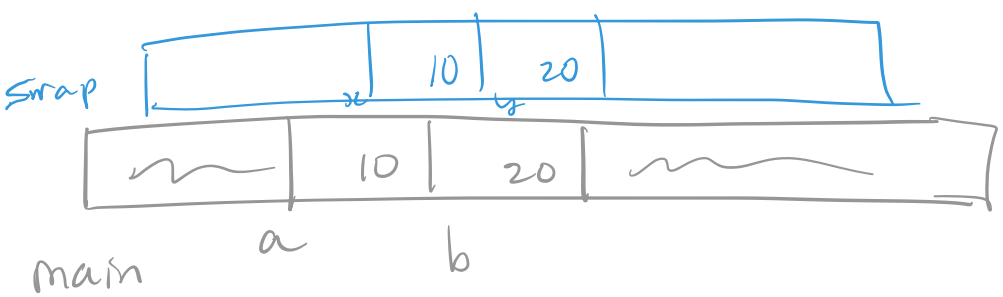
But that's confusing, so don't do it.

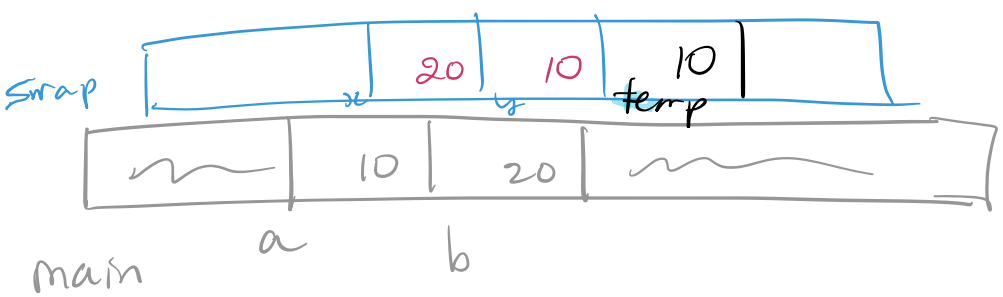
```
#include <iostream>
using namespace std;

void swap(int x, int y);
int main()
{
    int a=10, b=20;
    swap(a,b);
    cout << a << "_" << b;
}

void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = x;
}
```

```
#include <iostream>
using namespace std;

void swap(int x, int y);
int main()
{
    int a=10, b=20;
    swap(a,b);
    cout << a << "_" << b;
}

void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = x;
}
```

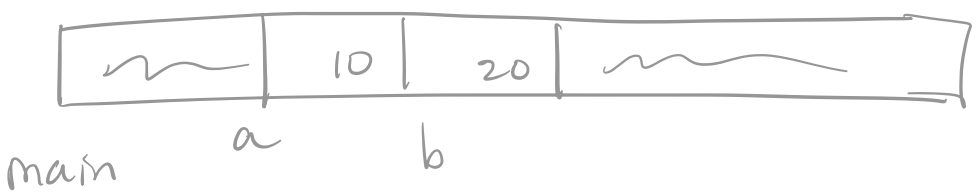
// Does not work
// as we would
// expect
// because
// default for
// simple data
// types
// (i.e. non-
// arrays)
// is pass-by-value

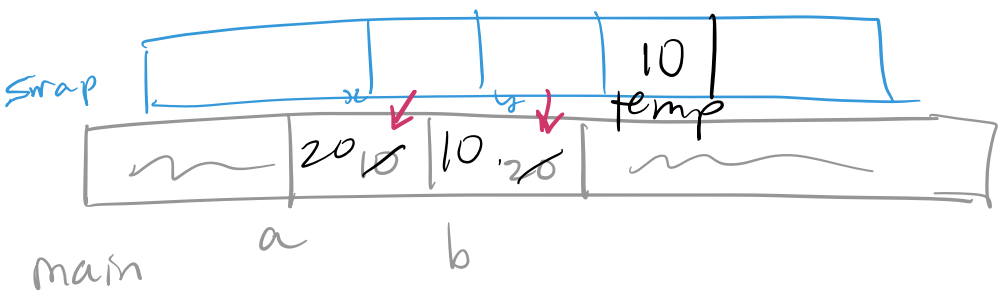

10
20
a
b
main


Swap
10
20
x
y
10
20
a
b
main


Swap
20
10
10
temp
10
20
a
b
main

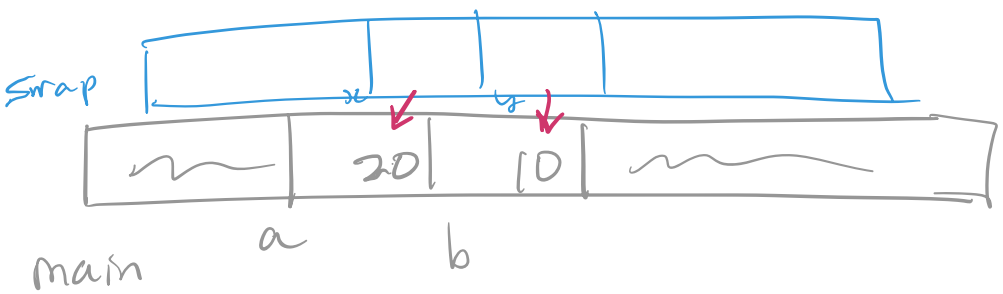
```
#include <iostream>
using namespace std;

void swap(int &x, int &y);
int main()
{
    int a=10, b=20;
    swap(a,b);
    cout << a << "_" << b;
}

void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = x;
}
```


10
20
a
b
main



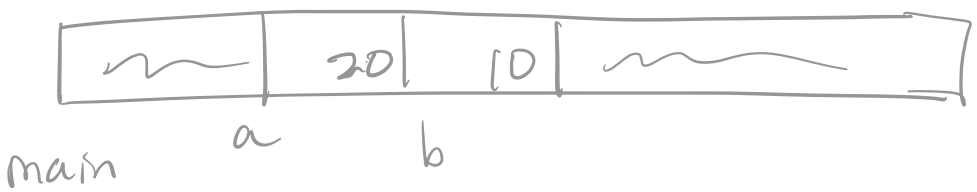
The & in front says:

"Send to the function the address of a and the address of b, so that the function's work will be on those variables and changes made will last after the function returns."



The & in front says:

"Send to the function the address of a and the address of b, so that the function's work will be on those variables and changes made will last after the function returns."



The & in front says:

"Send to the function the address of a and the address of b, so that the function's work will be on those variables and changes made will last after the function returns."

In pass-by-reference, the argument must be a variable of the correct type.

```
void print(int x)
{
    cout << x;
}

void printAndIncrement(int &x)
{
    cout << x++;
}
```

In a calling function (say main) we could execute this:

```
print(y+12);
```

but not this:

```
printAndIncrement(y+12);
```

Nor this?

```
printAndIncrement(12);
```

Nor:

```
printAndIncrement(12.0);
```

What about

```
float b = 12.1;
print(b);
```

?

Arrays as parameters

Arrays are always passed by reference

... so Lab 4 is in error when it says you would see something different (ie not sorted) if you leave off the & in the parameter list.

Arrays as parameters

Arrays are always passed by reference

... so Lab 4 is in error when it says you would see something different (ie not sorted) if you leave off the & in the parameter list.

```
void swapLeft(float arr[], int index,
              int sz)
{
    if (index < 1 || index > sz-1)
    {
        return;
    }
    float temp = arr[index];
    arr[index] = arr[index-1];
    arr[index-1] = temp;
}
```

NB: the & when used in this way is part of the prototype and the definition

prototype ← "declaration"

definition ← code

Not at the call

```
void  inc(int &i);
```

declaration

```
int main()
{
    :
    inc(numClients);
    :
}
```

call

```
void  inc(int &i)
{
    i++;
}
```

definition

Pass-by-reference uses

A function call can only return one value. We can use pass-by-reference to "get back" several values at once.

```
void getCoords(int &x, int &y);

int main()
{
    int x1, y1, x2, y2;
    getCoords(x1, y1);
    getCoords(x2, y2);
    cout << x1 << " " << y1 << endl;
}

void getCoords(int &x, int &y)
{
    cout << "Enter x coordinate: ";
    cin >> x;
    cout << "Enter y coordinate: ";
    cin >> y;
}
```

What would happen if the & had been left off?

Use pass-by-reference only when you need to change the value.

- unexpected "side effects" can lead to buggy code.