IIT Bombay

Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Pointers in Function Calls

Quick Recap of Relevant Topics

IIT Bombay

- Basic programming constructs
- Pointer data type in C++
- "Address of" operator in C++
- "Content of" operator in C++

Used "address of" and "content of" operators within the same function

Overview of This Lecture

IIT Bombay

- Using pointers across functions
  - Pointers as parameters to functions
  - Comparison with call-by-reference
  - Returning pointers from functions

Recap: Memory, Addresses and Pointers

IIT Bombay

- Main memory is a sequence of storage locations
- Each location contains a value (content) and has a unique address
- A pointer is an address of a location allocated in main memory to store a value
- Pointer valued variables can store addresses of memory locations

Recap: Function Calls

IIT Bombay

- Passing parameters to functions

  Call-by-value

  Call-by-reference
- Use of activation records in call stack to manage local variables, passing of parameters and also flow of control
- All local variables of a function allocated space in the activation record of the function

Can We Pass Pointers as Function Parameters?

- Why not?
- Should it be call-by-value or call-by-reference?
  - Mostly call-by-value for our purposes
  - However, C++ allows passing references to pointers as well
    - References to pointer-valued (int *, char *, …) variables treated in same way as references to variables of other basic data types (int, char, …)

Pointers as Function Parameters

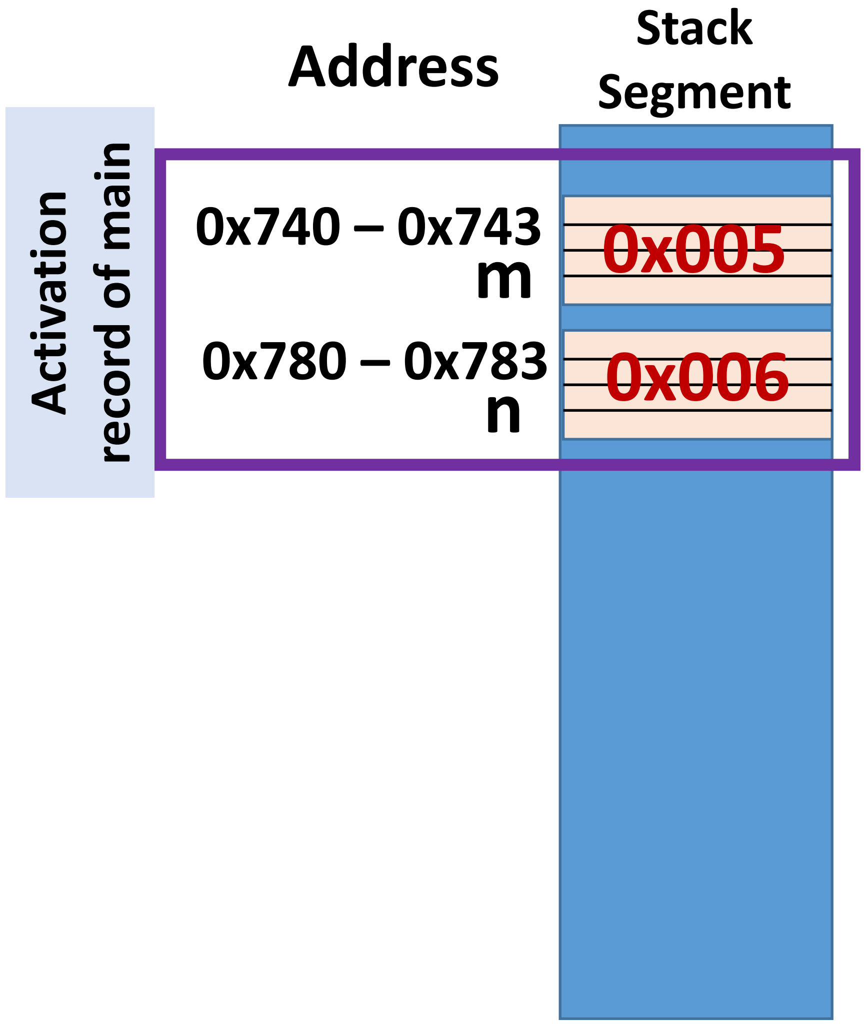
IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY);

int main()
{  int m; int n;
   cout << “Give m and n: “;
   cin >> m >> n;
   swapByPtr(&m, &n);
   cout << “m: “ << m << endl;
   cout << “n: “ << n << endl;
   return 0;
}
```

```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

Pointers as Function Parameters

```
void swapByPtr(int *ptrX, int *ptrY);

int main()
{  int m; int n;
   cout << “Give m and n: “;
   cin >> m >> n;
   swapByPtr(&m, &n);
   cout << “m: “ << m << endl;
   cout << “n: “ << n << endl;
   return 0;
}
```

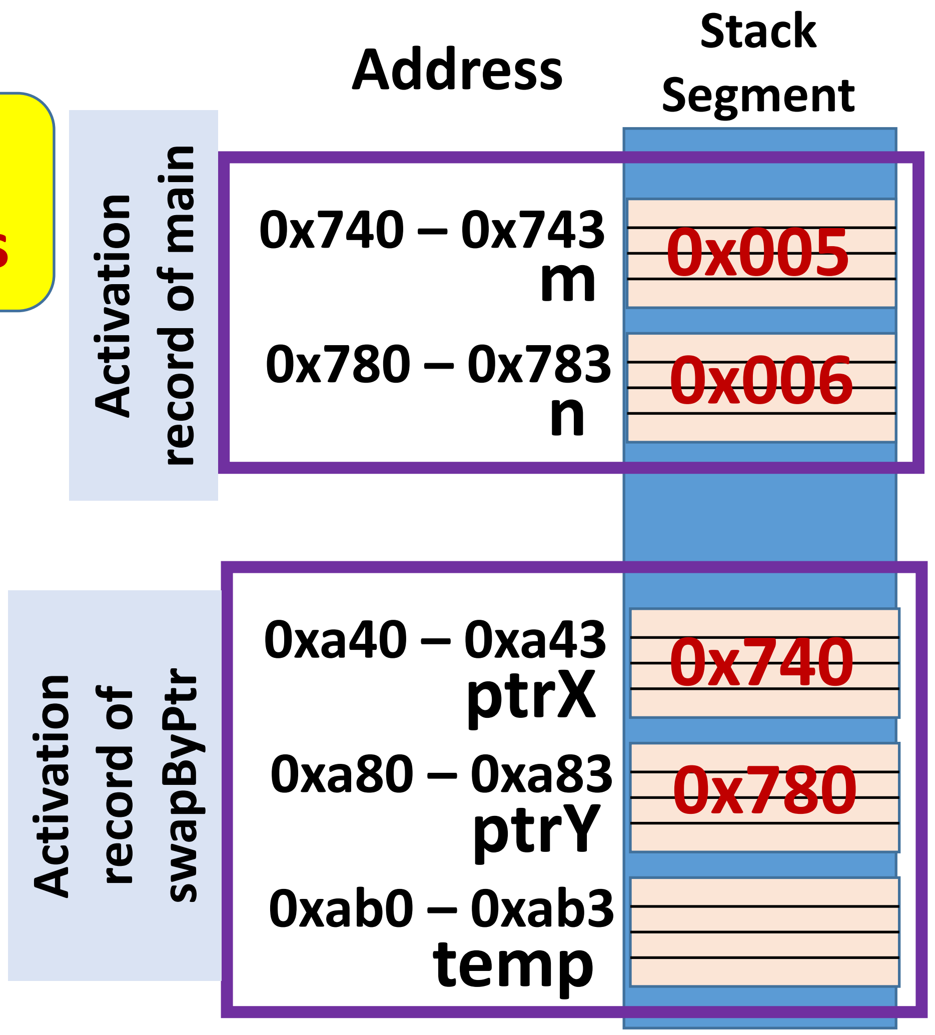


Pointers as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY);

int main(

{  int m; int n;
   cout << “Give          d n: “;
   cin >> m >> r
   swapByPtr(&m, &n);
   cout << “m: “ << m << endl;
   cout << “n: “ << n << endl;
   return 0;
}
```

Parameters are addresses.
“call-by-value” with addresses

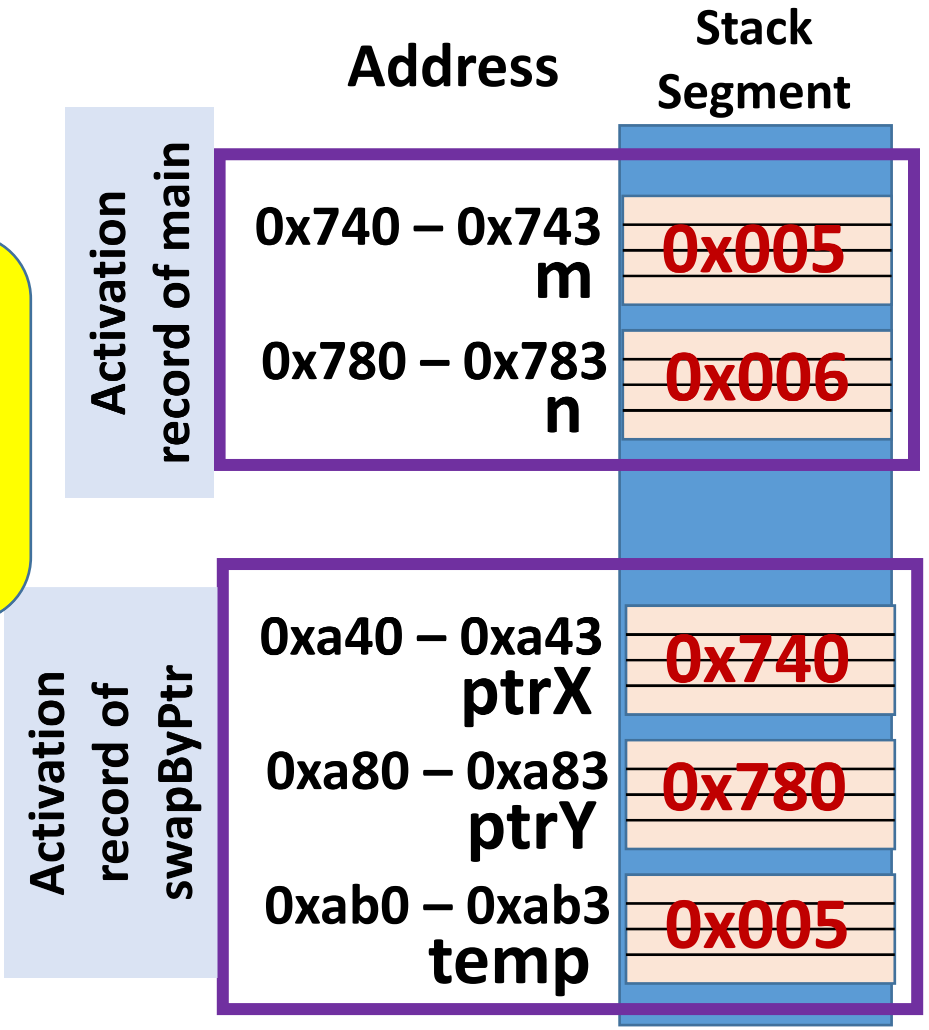


Pointers as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

Accessing contents of memory location in activation record of main from swapByPtr

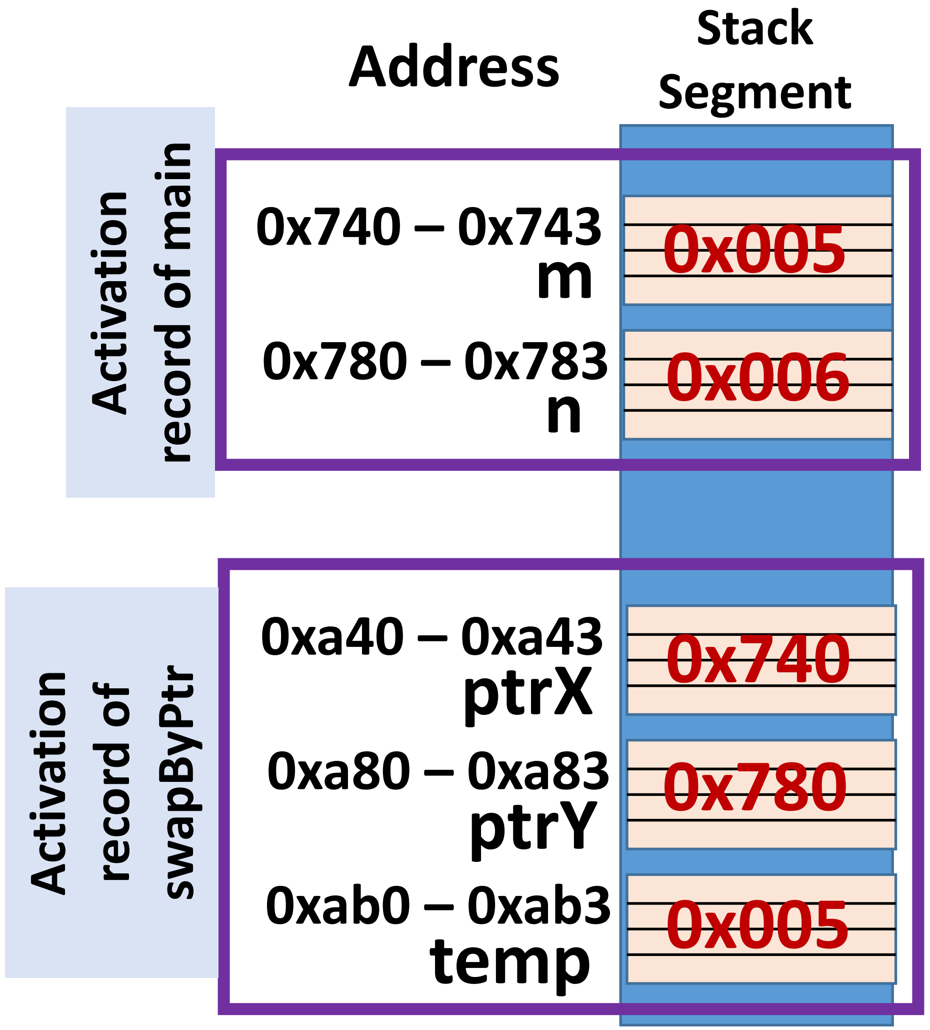


Pointers as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

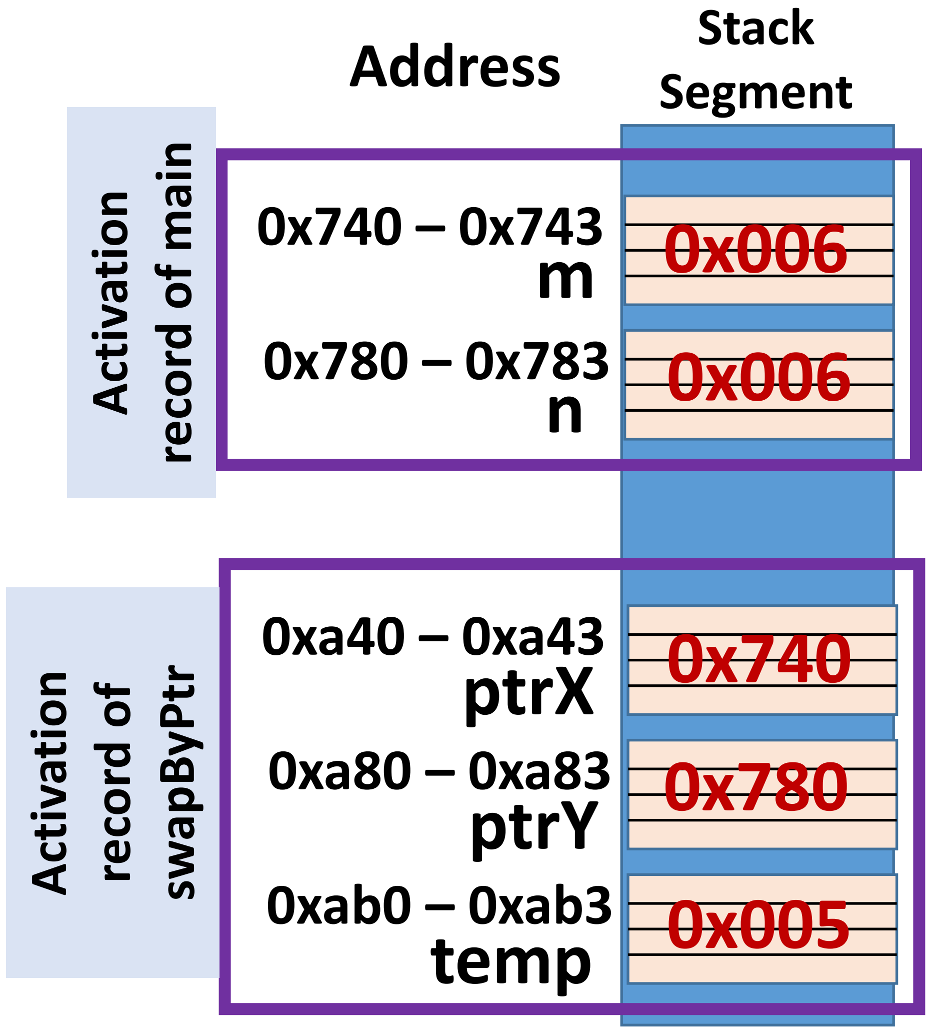
Update contents of memory at address 0x740 with contents of memory at address 0x780



Pointers as Function Parameters

```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

Update m in activation record of main with value of n in activation record of main

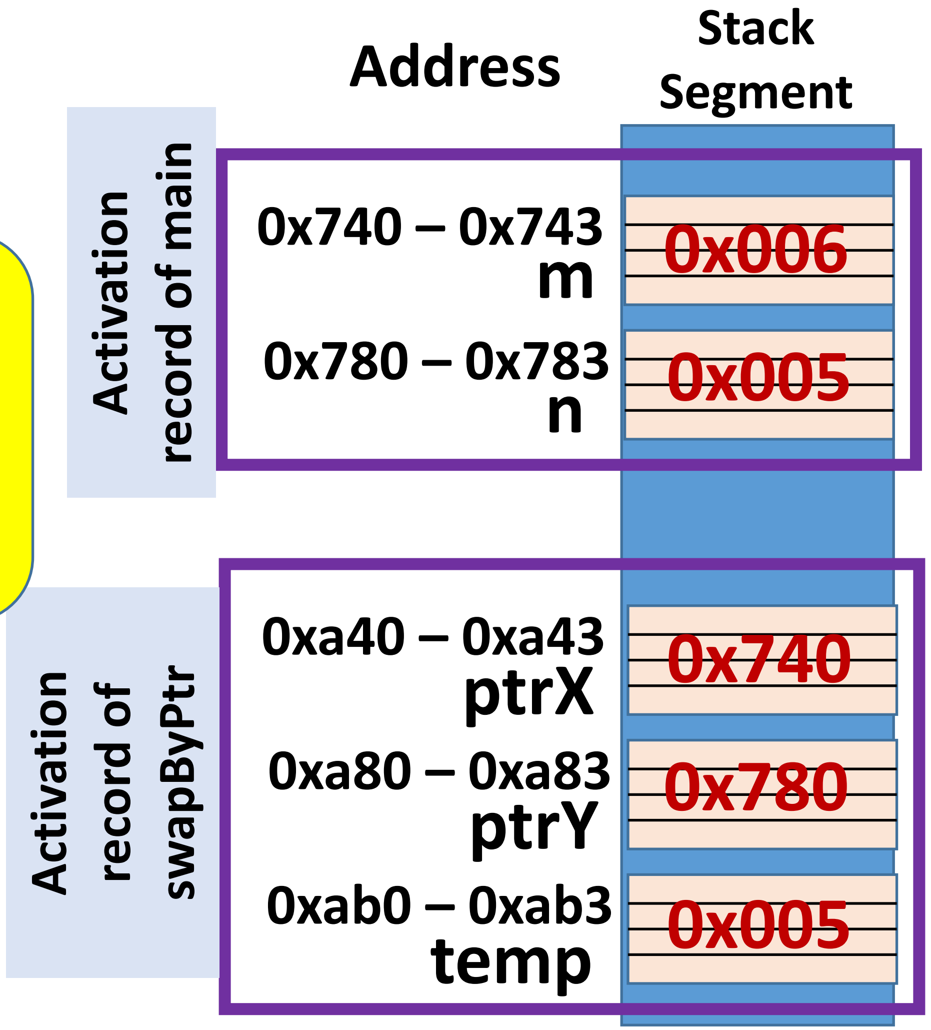


Pointers as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```

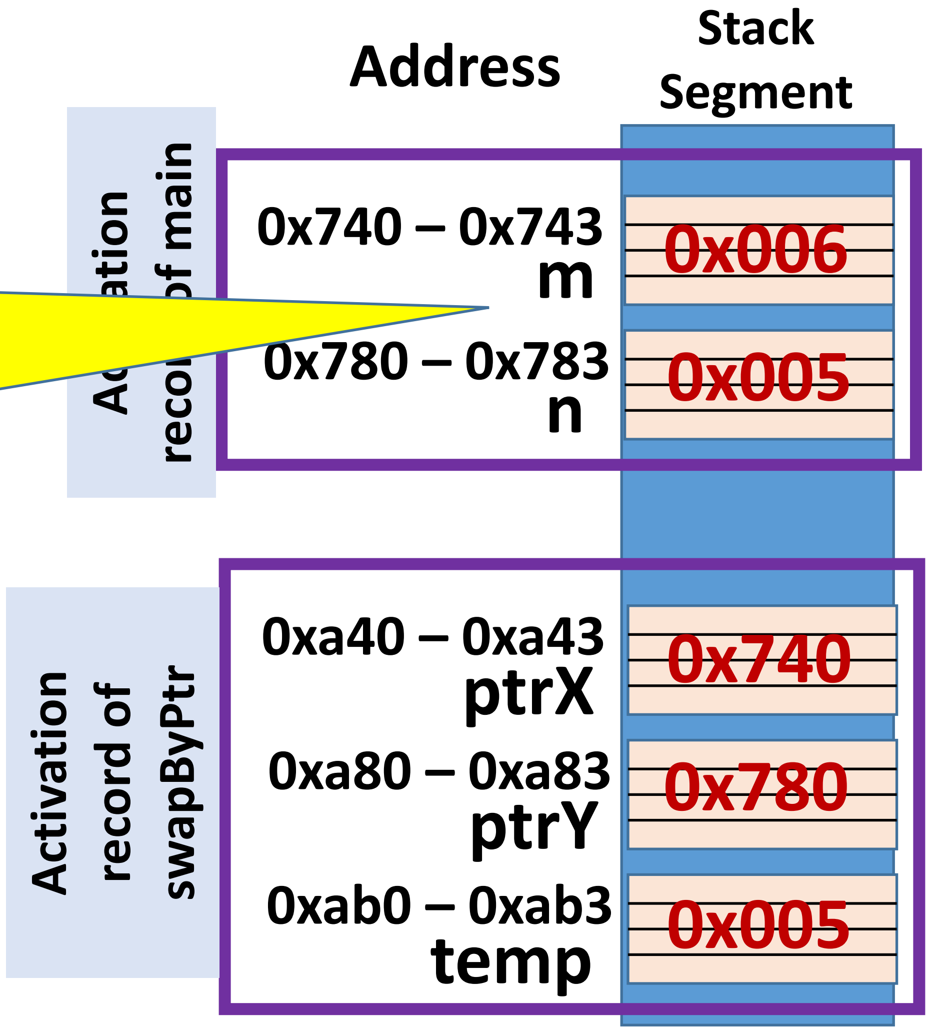
Accessing contents of memory location in activation record of main from swapByPtr



Pointers as Function Parameters

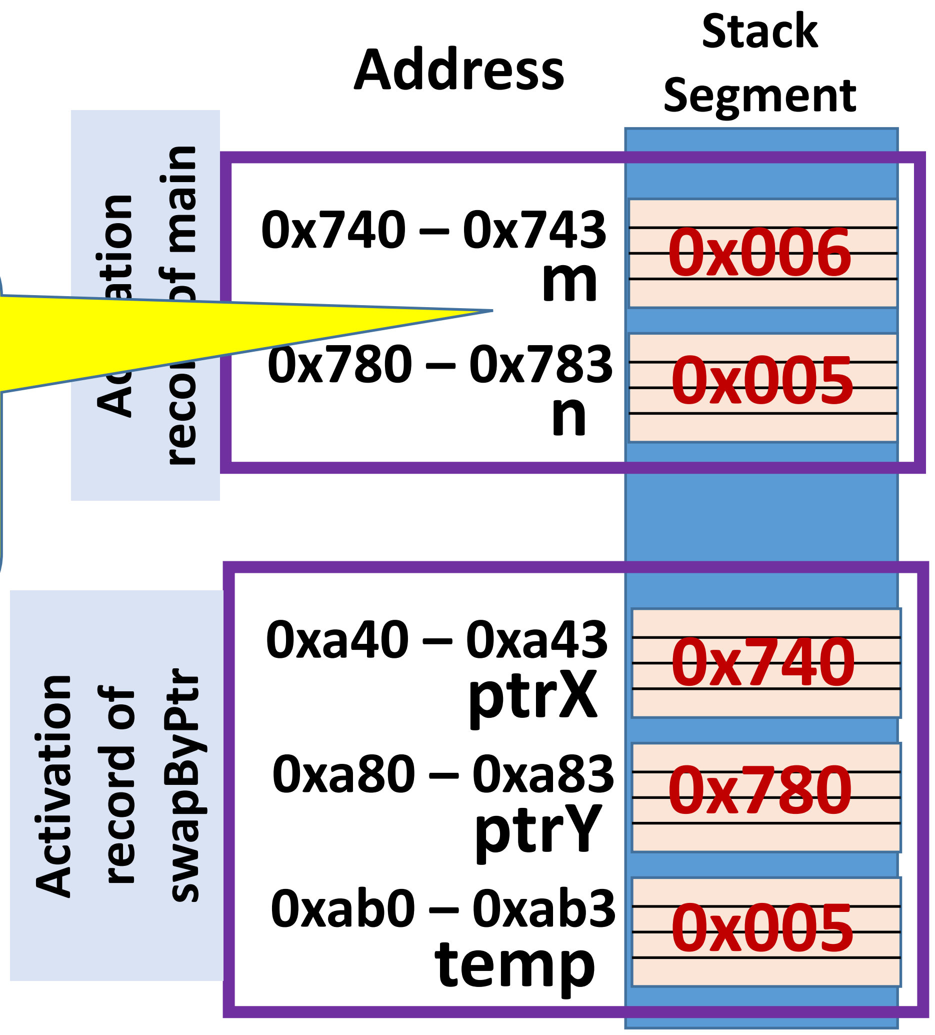
```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
   return;
}
```

Contents of m and n in activation record of main are swapped !!!



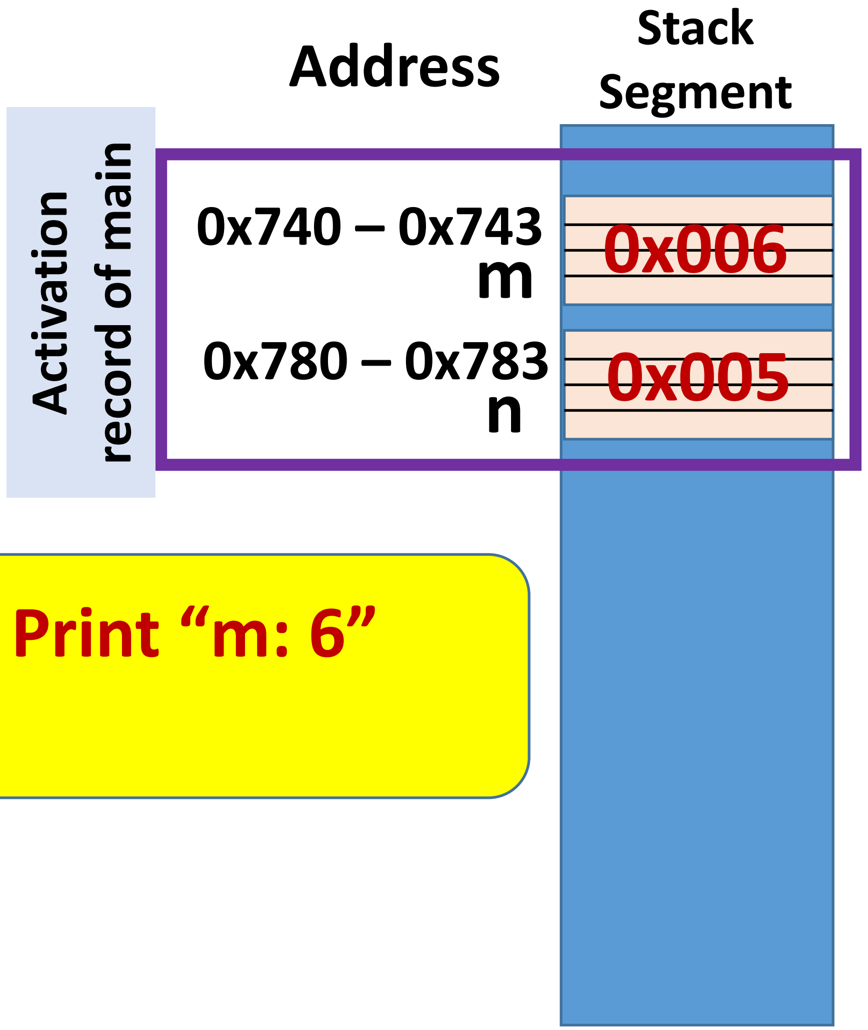
Pointers as Function Parameters

```
void swapByPtr(int *ptrX, int *ptrY)
{
  int temp;
  temp = *ptrX;
  *ptrX = *ptrY;
  *ptrY = temp;
  return;
}
```



Pointers as Function Parameters

```
void swapByPtr(int *ptrX, int *ptrY);

int main()
{  int m; int n;
   cout << "Give m and n: ";
   cin >> m >> n;
   swapByPtr(&m, &n);
   cout << "m: " << m << endl;
   cout << "n: " << n << endl;
   return 0;
}
```

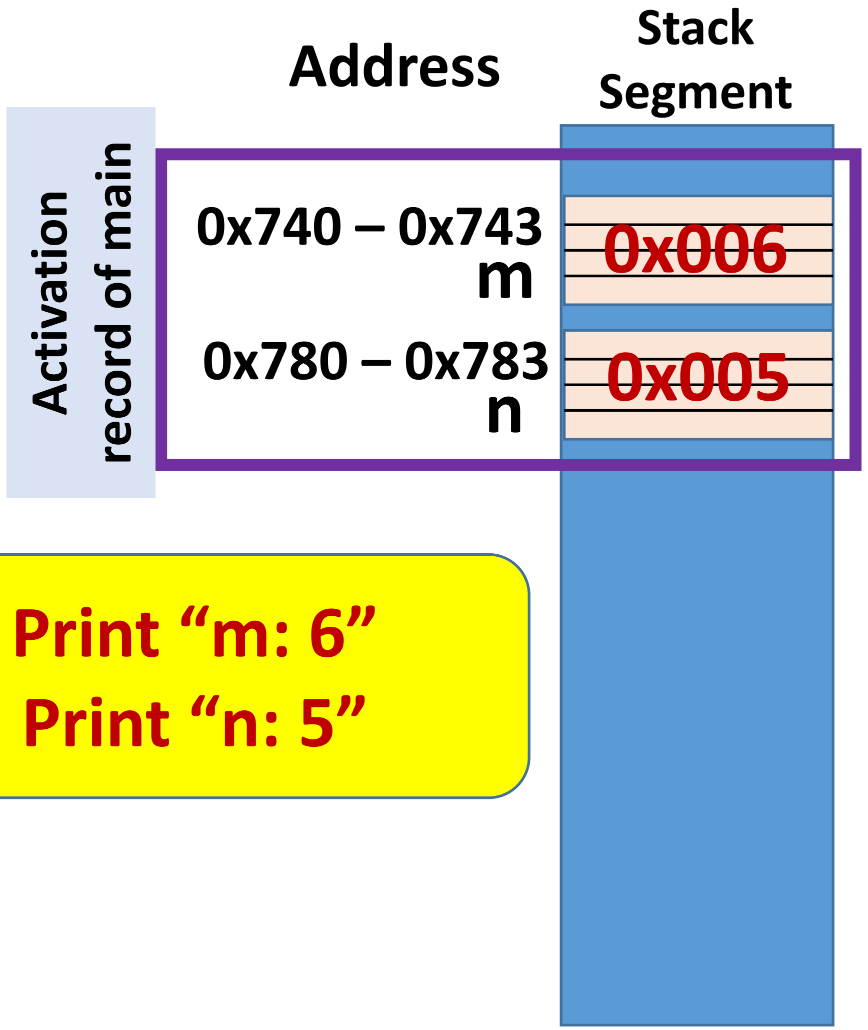


Print "m: 6"

Pointers as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY);

int main()
{  int m; int n;
   cout << “Give m and n: “;
   cin >> m >> n;
   swapByPtr(&m, &n);
   cout << “m: “ << m << endl;
   cout << “n: “ << n << endl;
   return 0;
}
```



Print “m: 6”
Print “n: 5”

Moral Of The Story

- By passing pointers as function parameters, callee (swapByPtr) gets access to local variables of caller (main)
- Another way to share variables between caller and callee
  - Passing parameters by reference also accomplishes this
  - In fact, when we pass parameters by reference in C++, after compilation, pointers to parameters are actually passed
    - Some more book-keeping done in call-by-reference
    - Pointers behind the scenes
    - Saves us some untidy coding !!!

Pointers and References as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY);
void swapByRef(int &X, int &Y);
int main()
{  int m; int n;
   cout << "Give m, n: "; cin >> m >> n;
   swapByPtr(&m, &n);
   swapByRef(m, n);
   cout << m << " " << n << endl;
   return 0;
}
```

```
void swapByPtr(int *ptrX, int *ptrY)
{  int temp;
   temp = *ptrX;   *ptrX = *ptrY;
  *ptrY = temp;   return;
}
```

```
void swapByRef(int &X, int &Y)
{  int temp;
   temp = X;  X = Y;
   Y = temp; return;
}
```

Pointers and References as Function Parameters

```
void swapByPtr(int *ptrX, int *ptrY);

int ma

{  int m; in
   cout << "Giv   m, n: "; cin >> m >> n;
   swapByPtr(&m, &n);
   swapByRef(m, n);
   cout << m << " " << n << endl;
   return 0;
}
```

Note how pointers are passed

```
void swapByPtr(int *ptrX, int *ptrY)
{  int temp;
   temp = *ptrX;   *ptrX = *ptrY;
   *ptrY = temp;   return;
}
```

```
void swapByRef(int &X, int &Y)
{  int temp;
   temp = X;  X = Y;
   Y = temp; return;
}
```

Pointers and References as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY);
void swapByRef(int &X, int &Y);

{ int r
  cout <<            m, n: “; cin >> m >> n;
  swapByPtr(    n, &n);
  swapByRef(m, n);
  cout << m << “ “ << n << endl;
  return 0;
}
```

Compare with how references are passed

```
void swapByPtr(int *ptrX, int *ptrY)
{  int temp;
   temp = *ptrX;  *ptrX = *ptrY;
  *ptrY = temp;   return;
}
```

```
void swapByRef(int &X, int &Y)
{  int temp;
   temp = X;  X = Y;
   Y = temp; return;
}
```

Pointers and References as Function Parameters

IIT Bombay

Think of swapByPtr as how the compiler implements swapByRef

Isn't swapByRef cleaner to use?

```
  swapByPtr(&m, &n);
  swapByRef(m, n);
  cout << m << " " << n << endl;
  return 0;
}
```

```
void swapByPtr(int *ptrX, int *ptrY)
{  int temp;
   temp = *ptrX;  *ptrX = *ptrY;
  *ptrY = temp;   return;
}
```

```
void swapByRef(int &X, int &Y)
{  int temp;
   temp = X;  X = Y;
   Y = temp; return;
}
```

Pointers and References as Function Parameters

IIT Bombay

```
void swapByPtr(int *ptrX, int *ptrY);
void swapByRef(int &X, int &Y);
int main()
{  int m; int n;
   cout << “Give m, n: “; cin >> m >> n;
   swapByPtr(&m, &n);
   swapByRef(m, n);
   cout << m << “ “ << n << endl;
   return 0;
}
```

```
void swapByPtr(int *ptrX, int *ptrY)
{  int temp;
   temp = *ptrX;  *ptrX = *ptrY;
  *ptrY = temp;   return;
}
```

```
void swapByRef(int &X, int &Y)
{  int temp;
   temp = X;  X = Y;
   Y = temp; return;
}
```

Can a Function Return a Pointer?

IIT Bombay

- Most certainly!
- Care needed so that the returned pointer does not point to a location in activation record of the function
    - Activation record freed when a function returns
    - Dereferencing an address in the freed activation record will cause program to crash

Function Returning A Pointer

```
int *myFunc(int *ptrB);
int main()
{
   int * a, b;
   cout << “Give b: “; cin >> b;
   a = myFunc(&b);
   cout << “a is: “ << *a << endl;
   return 0;
}
```

```
int * myFunc(int *ptrB)
{
   int a;
   a = (*ptrB) * (*ptrB);
   return (&a);
}
```

Function Returning A Pointer

```
int *myFunc(int *ptrB);
int main()
{
   int * a, b;
   cout << "Give b: "; cin >> b;
   a = myFunc(&b);
   cout << "a is: " << *a << endl;
   return 0;
}
```

```
int * myFunc(int *ptrB)
{
   int a;
   a = (*ptrB) * (*ptrB);
   return (&a);
}
```

Local variable in activation record of myFunc

Address of local variable in activation record of myFunc

Function Returning A Pointer

```
int *myFunc(int *ptrB);
int r
{
   in
   cout       Give b: "; cin >> b;
   a = myFunc(&b);
   cout << "a is: " << *a << endl;
   return 0;
}
```

Address of local variable in non-existent activation record of myFunc: BAD ADDRESS

```
int * myFunc(int *ptrB)
{

   return (&a);
}
```

Dereferencing a BAD ADDRESS

Another Function Returning A Pointer

```
int *myFunc(int *ptrB);
int main()
{
   int * a, b;
   cout << "Give b: "; cin >> b;
   a = myFunc(&b);
   cout << "a is: " << *a << endl;
   return 0;
}
```

```
int * myFunc(int *ptrB)
{
   int a;
   a = (*ptrB) * (*ptrB);
   *ptrB = a;
   return (ptrB);
}
```

Another Function Returning A Pointer

```
int *myFunc(int *ptrB);
int main()
{
   int * a, b;
   cout << “Give b: “; cin >> b;
   a = myFunc(&b);
   cout << “a is: “ << *a << endl;
   [illegible]
}
```

```
int * myFunc(int *ptrB)
{
   int a;
   a = (*ptrB) * (*ptrB);
   *ptrB = a;
   return (ptrB);
}
```

Local variable in activation record of myFunc

Address of variable in activation record of main

Another Function Returning A Pointer

```
int *myFunc(int *ptrB);
int main()
{
   int * a, b;
   cout << “Give b: “; cin >> b;
   a = myFunc(&b);
   cout << “a is: “ << *a << endl;
   return 0;
}
```

```
int * myFunc(int *ptrB)


   a = (*ptrB) * (*ptrB);

   return (ptrB);
}
```

Address of variable in activation record of main

Dereferencing a legitimate address

Summary

IIT Bombay

- Pointers (addresses) as parameters to functions
- Comparison with call-by-reference parameter passing
- Caveats when returning pointers from functions