Lecture 3 – Functions Revision

Revision – Lecture 1

- Address Operator
- Dereference Operator
- Null Reference Exception
- Dangling Pointer
- Illegal Memory Access with Dangling Pointer
- Why Size of all pointers is 4 Bytes?

Revision – Lecture 2

- Class Activity – Behavior of Code Snippets
- Motivation behind Dynamic Allocation
- Functionality of Operator new
- Data Access using
  - Subscript Operator, arr[i]
  - Offset Notation, *(arr+i)
- Memory Deallocation, delete Operator
- Memory Leakage

Revision – Lecture 2

- Arithmetic and Logical Operations with Pointers
  - Ptr++
  - Ptr--
  - Ptr+1
  - Ptr1 < Ptr2

Revision

- Assignment Discussion
- Lab Manual Discussion
  - Practice Problems
- Coding Standards

New/Delete Operator

- Int* ptr = New int; //allocates one integer on heap
- Int* ptr = New int[5]; //allocates 5 integers on heap
- Int* ptr = New int(5);//allocates one integer on heap and initializes it to value 5.
- Delete x; // Deallocates one integer
- Delete[] arr; //Deallocates an array

Class Activity 1

- Determine the behavior of following code. Identify exact problem(if any) and line.

```
int* x = new int[5]; //Suppose data in array is
//1,2,3 and so on.
X[3] = x[1]+x[2];
Cout<<x[3];
x = new int[10];
If(x) delete[] x;  // if(x != 0)
```

Class Activity 1 - Solution

- Determine the behavior of following code:

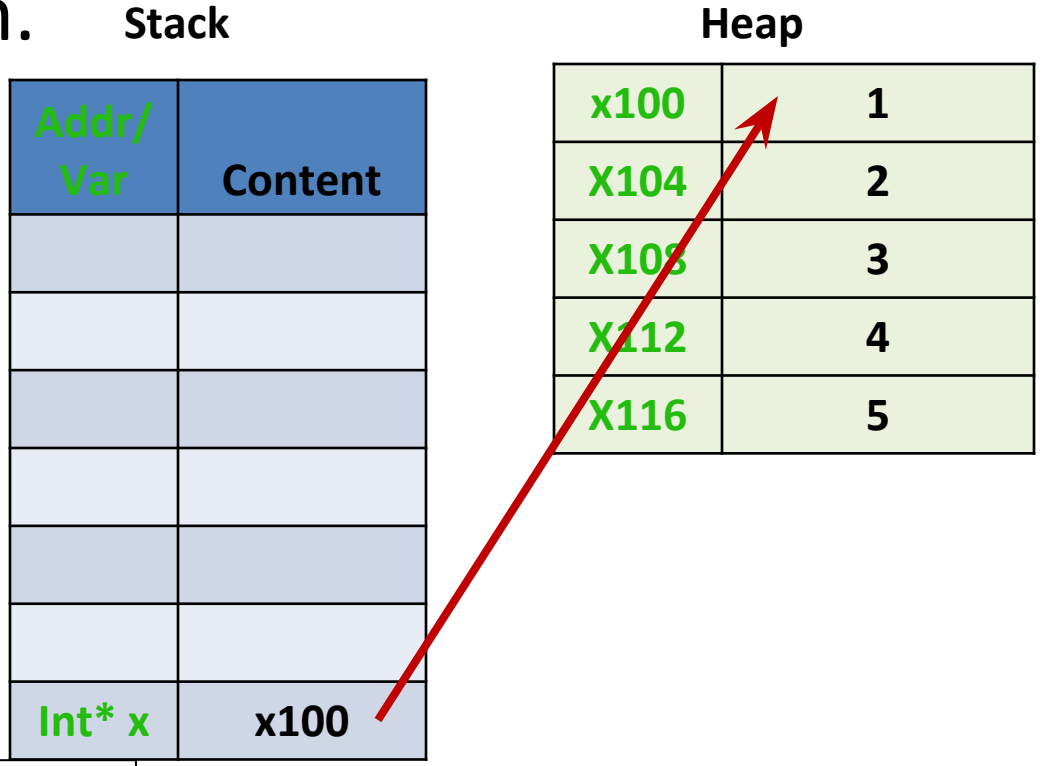
int* x = new int[5]; //Suppose data in array is //1,2,3 and so on.

X[3] = x[1]+x[2];

Cout<<x[3]

x = new int[10];

If(x) delete[] x;

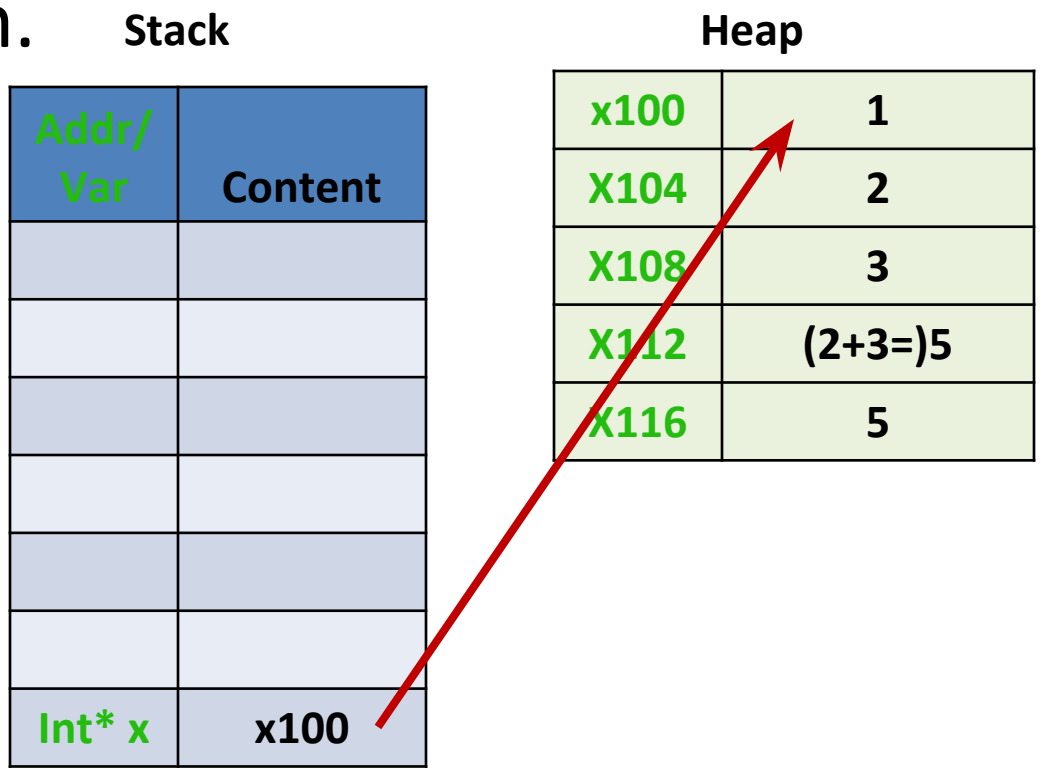


Allocated space is highlighted in green

Class Activity 1 - Solution

- Determine the behavior of following code:

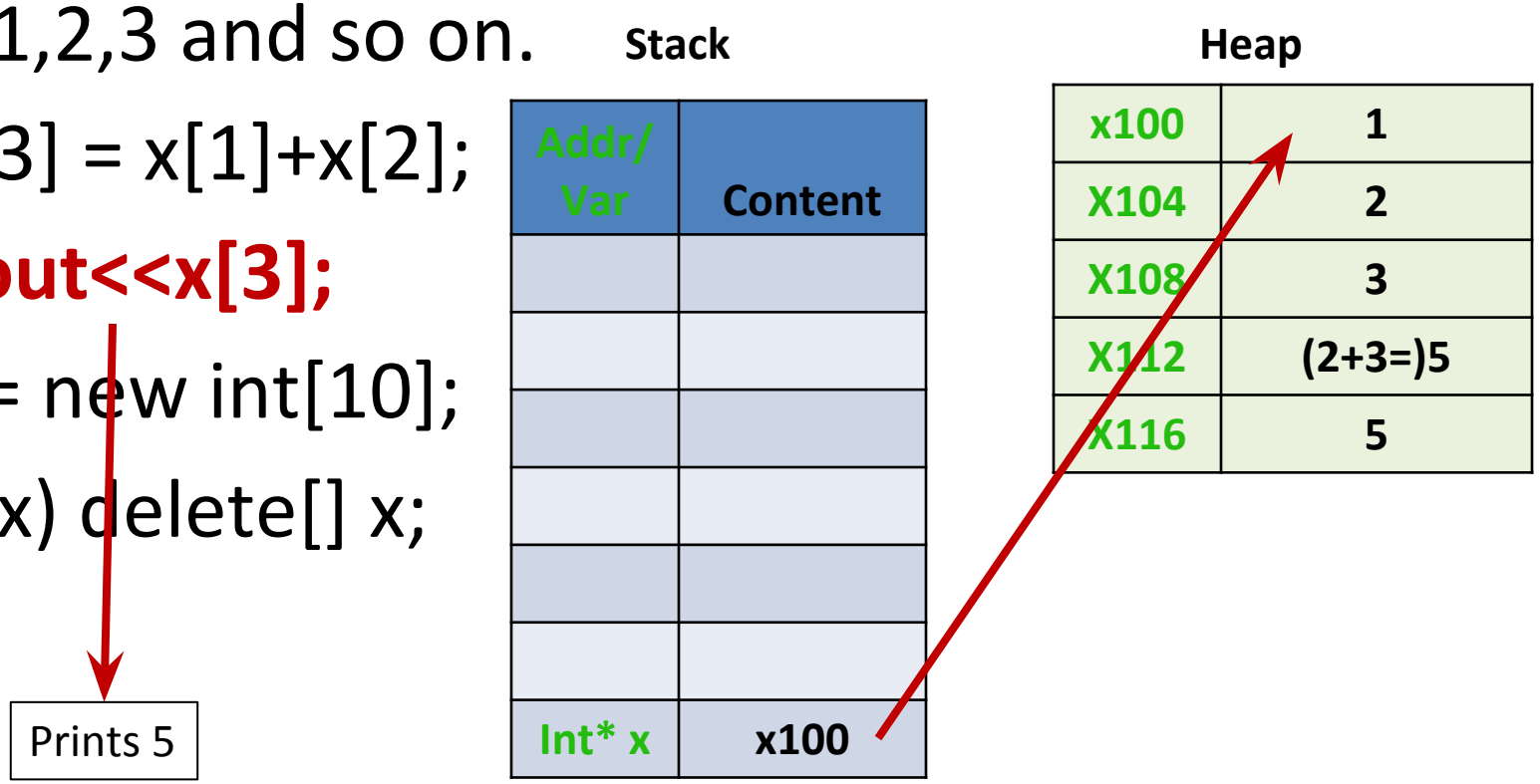
```
int* x = new int[5]; //Suppose data in array is
//1,2,3 and so on.
X[3] = x[1]+x[2];
Cout<<x[3]
x = new int[10];
If(x) delete[] x;
```



Class Activity 1 - Solution

- Determine the behavior of following code:

```
int* x = new int[5]; //Suppose data in array is
//1,2,3 and so on.
X[3] = x[1]+x[2];
Cout<<x[3];
x = new int[10];
If(x) delete[] x;
```

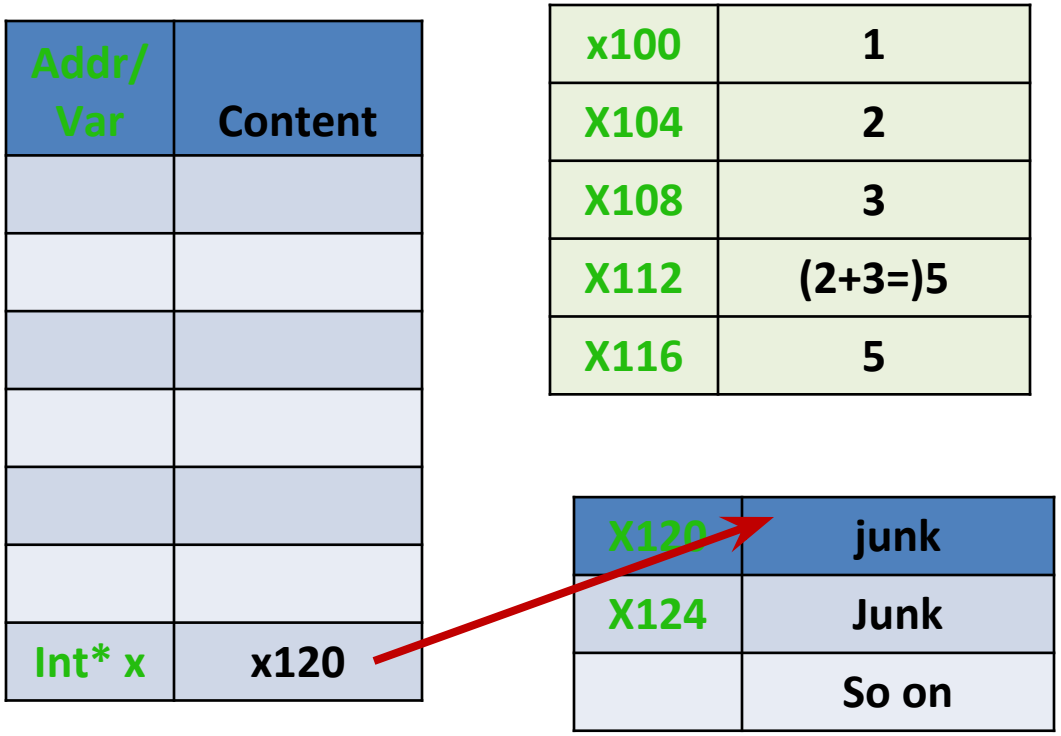


Class Activity 1 - Solution

- Determine the behavior of following code:

```
int* x = new int[5]; //Suppose data in array is
//1,2,3 and so on.
X[3] = x[1]+x[2];
Cout<<x[3];
x = new int[10];
If(x) delete[] x;
```

New memory allocated. Base address of prev. array Lost.



Class Activity 1 - Answer

- Determine the behavior of following code. Identify exact problem(if any) and line.

```
int* x = new int[5]; //Suppose data in array is //1,2,3 and so on.
X[3] = x[1]+x[2];
Cout<<x[3]
```

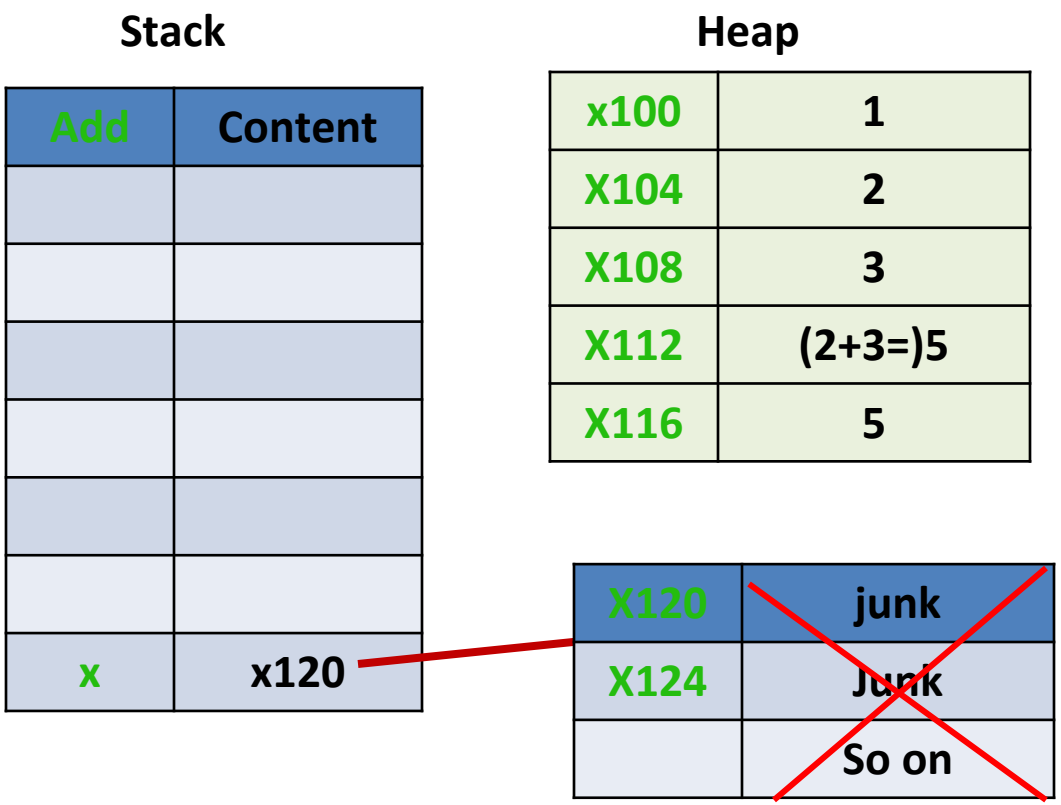
x = new int[10]; //Logical Error: Memory Leakage, array of 5 integers allocated above has been lost. (This is not an exception so the program will execute and terminate successfully.)

```
If(x) delete[] x;
```

Class Activity 1 – Solution (contd.)

- Determine the behavior of following code:

```
int* x = new int[5];
X[3] = x[1]+x[2];
Cout<<x[3]
x = new int[10];
If(x != 0){
 delete[] x;
//X = 0;}
```



2nd array deallocated.

Class Activity 2

- Determine the behavior of following code:

```
int* x = new int[5]; //Suppose it initializes the
//data to zero
Int* y = x;
X[3] = x[1]+x[2];
Delete[] x;
Cout<<y[3];
Y[3] = 999;
```

Class Activity 2 – Solution

- Determine the behavior of following code:

int* x = new int[5]; //Suppose it initializes the //data to zero

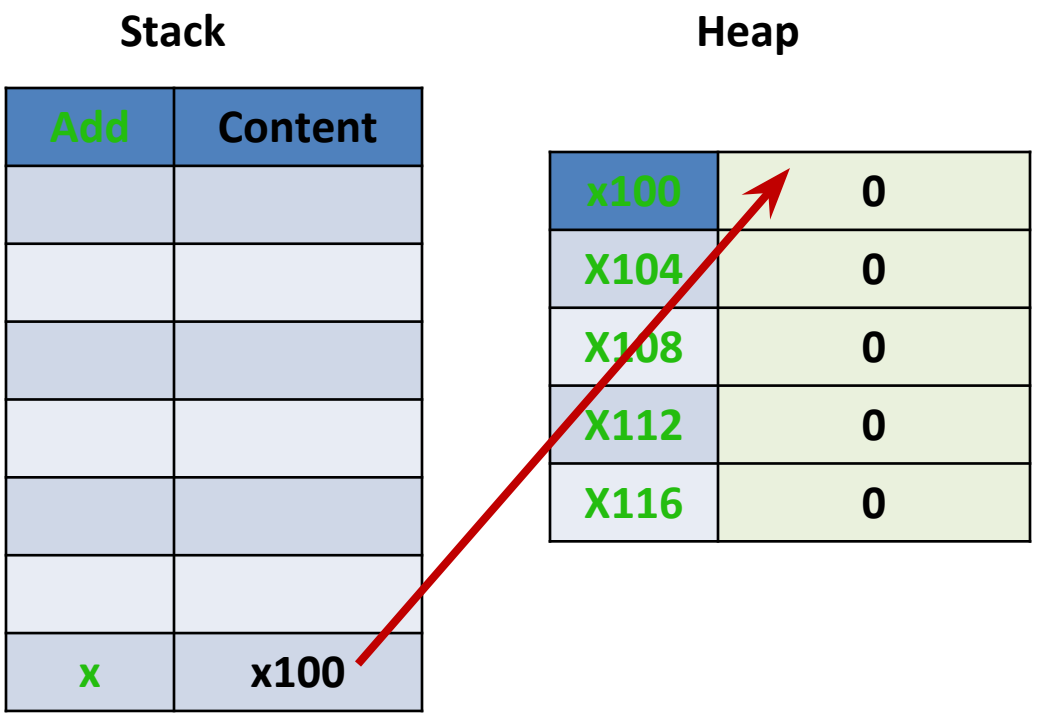
Int* y = x;

X[3] = x[1]+x[2];

Delete[] x;

Cout<<y[3];

Y[3] = 999;

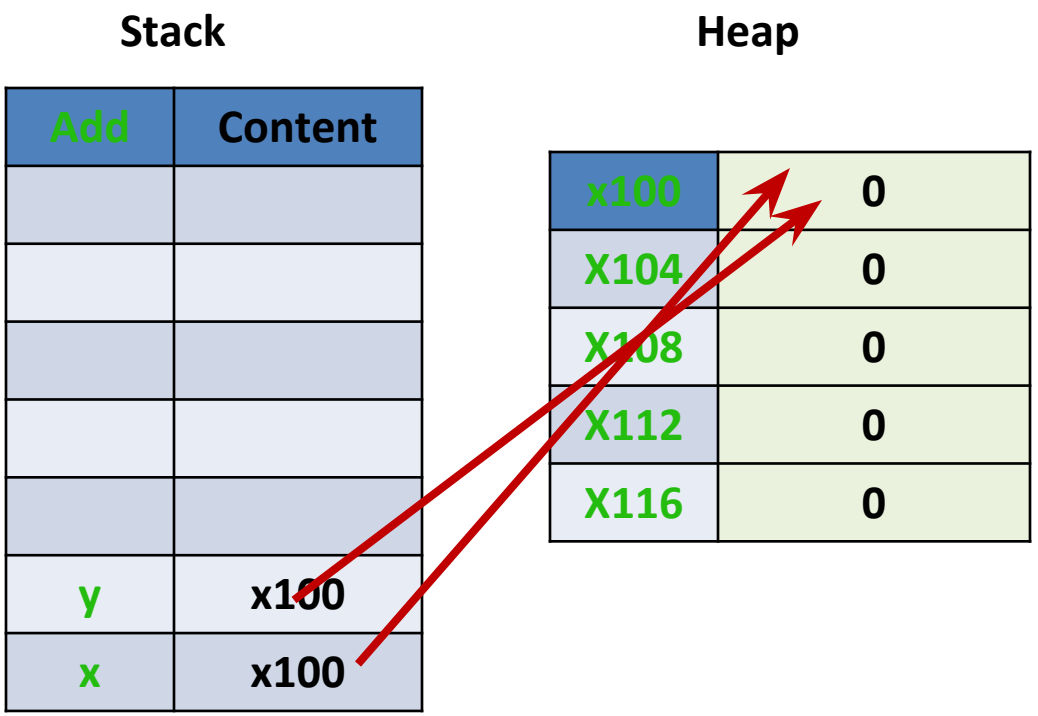


5 integers allocated on heap

Class Activity 2 – Solution

- Determine the behavior of following code:

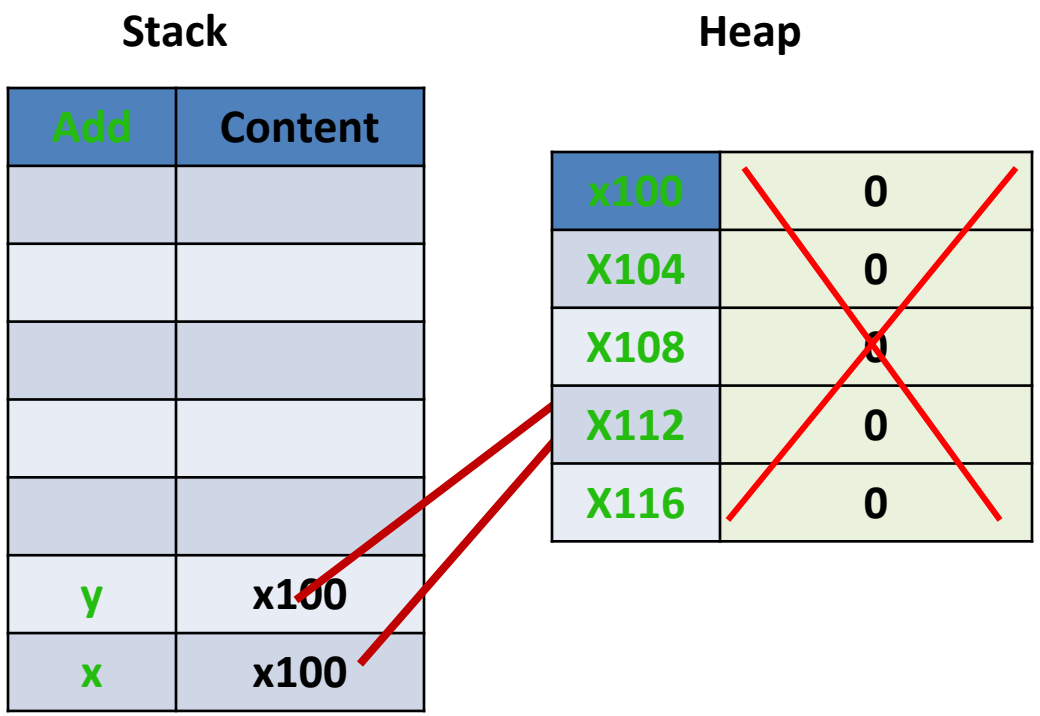
```
int* x = new int[5]; //Suppose it initializes the
//data to zero
Int* y = x;
X[3] = x[1]+x[2];
Delete[] x;
Cout<<y[3];
Y[3] = 999;
```



Class Activity 2 – Solution

- Determine the behavior of following code:

```
int* x = new int[5]; //Suppose it initializes the
//data to zero
Int* y = x;
X[3] = x[1]+x[2];
Delete[] x;
Cout<<y[3];
Y[3] = 999;
```

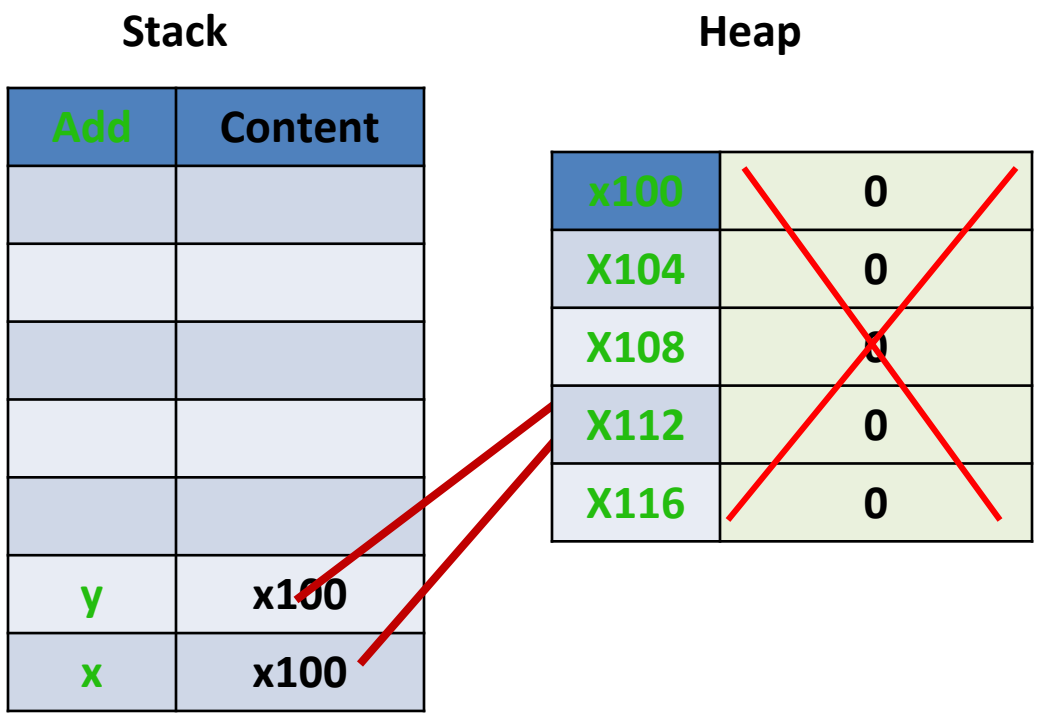


Array of baseAddrss x deallocated

Class Activity 2 – Solution

- Determine the behavior of following code:

```
int* x = new int[5]; //Suppose it initializes the
//data to zero
Int* y = x;
X[3] = x[1]+x[2];
Delete[] x;
Cout<<y[3];
Y[3] = 999;
```



Y is dangling pointer, trying illegal memory access.

Class Activity 2 - Answer

- Determine the behavior of following code:

```
int* x = new int[5]; //Suppose it initializes the //data to zero
Int* y = x;
X[3] = x[1]+x[2];
Delete[] x;
Cout<<y[3]; //Error: y is Dangling Pointer, trying illegal memory access
Y[3] = 999;
```

Class Activity 3

```
Int a = 10;
Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5
*aptr = a;
Int* bptr = aptr;
Delete aptr;
*bptr = a*5;
Cout<<*aptr<<*bptr<<endl;
```

Class Activity 3

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

*aptr = a;

Int* bptr = aptr;

Delete aptr;

*bptr = a*5;

Cout<<*aptr

<<*bptr<<endl;

Stack

Add	Content
a	10

Class Activity 3

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

*aptr = a;

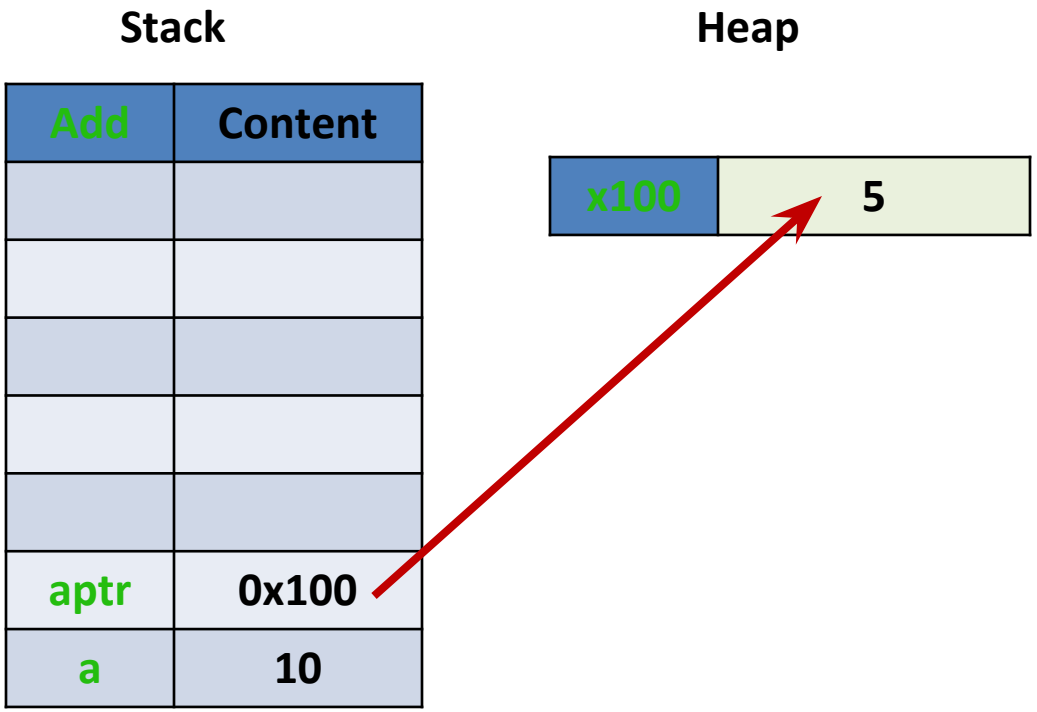
Int* bptr = aptr;

Delete aptr;

*bptr = a*5;

Cout<<*aptr

<<*bptr<<endl;



Allocated space is highlighted in green.

Class Activity 3

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

***aptr = a;**

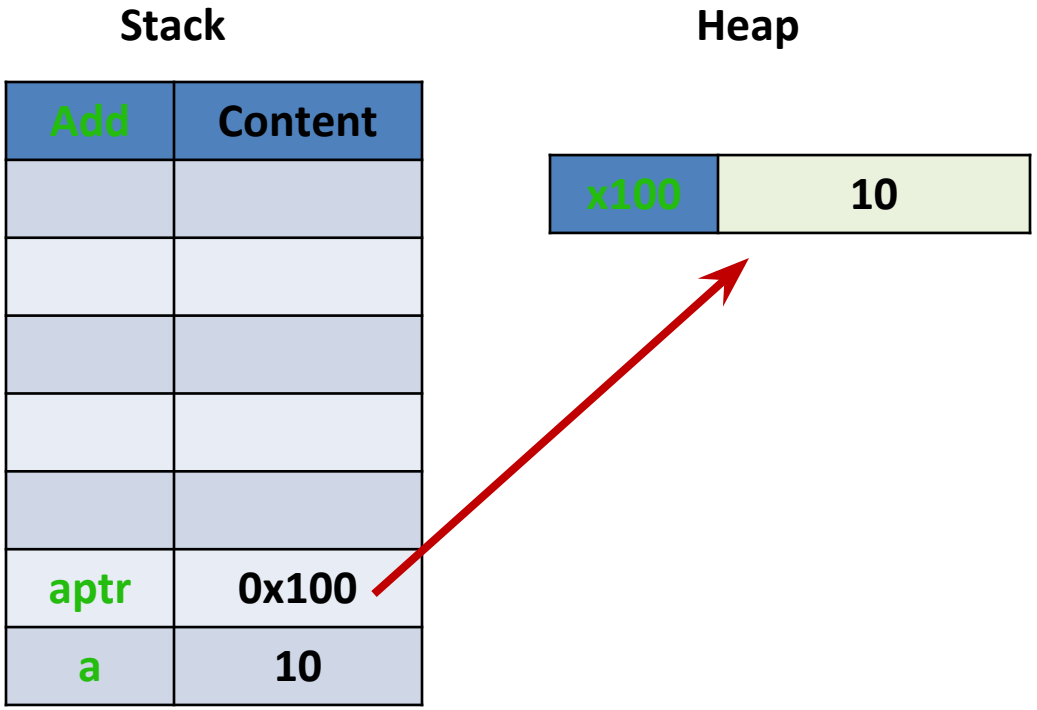
Int* bptr = aptr;

Delete aptr;

*bptr = a*5;

Cout<<*aptr

<<*bptr<<endl;



Content of a (i.e. 10) copied at location x100

Class Activity 3

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

*aptr = a;

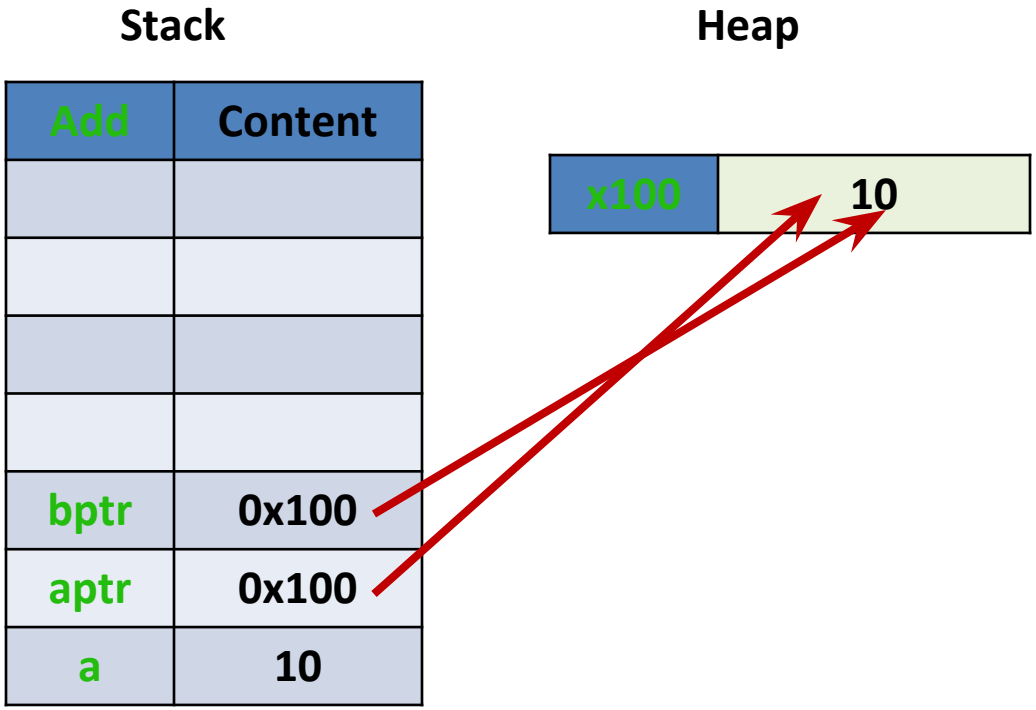
Int* bptr = aptr;

Delete aptr;

*bptr = a*5;

Cout<<*aptr

<<*bptr<<endl;



Class Activity 3

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

*aptr = a;

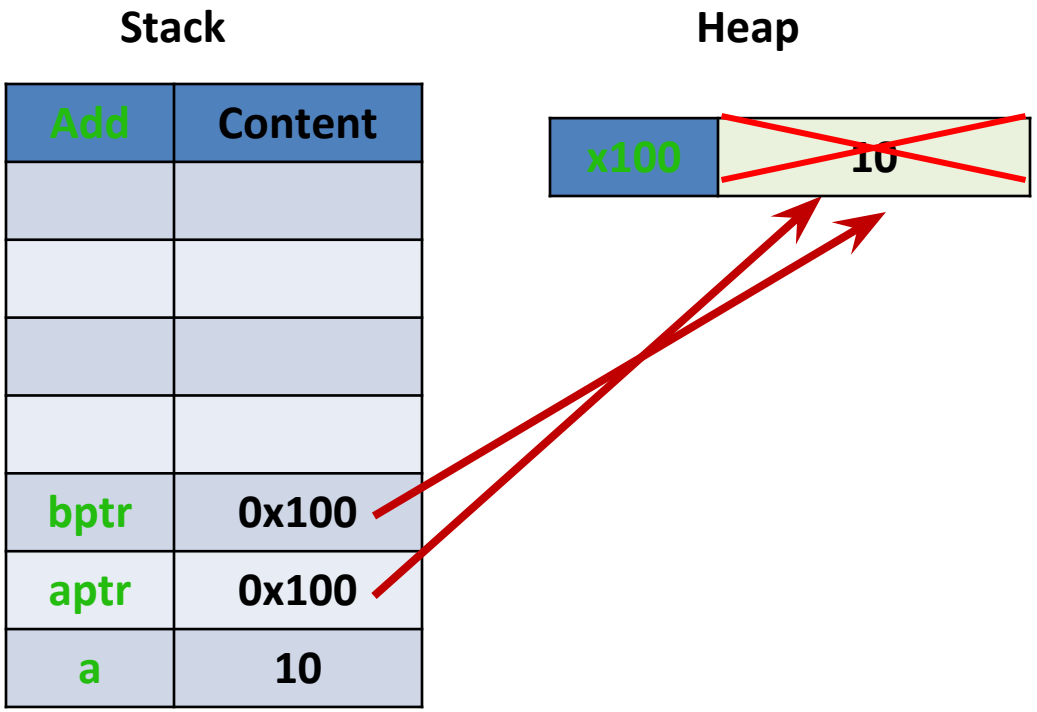
Int* bptr = aptr;

Delete aptr;

*bptr = a*5;

Cout<<*aptr

<<*bptr<<endl;



Memory deallocated

Class Activity 3

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

*aptr = a;

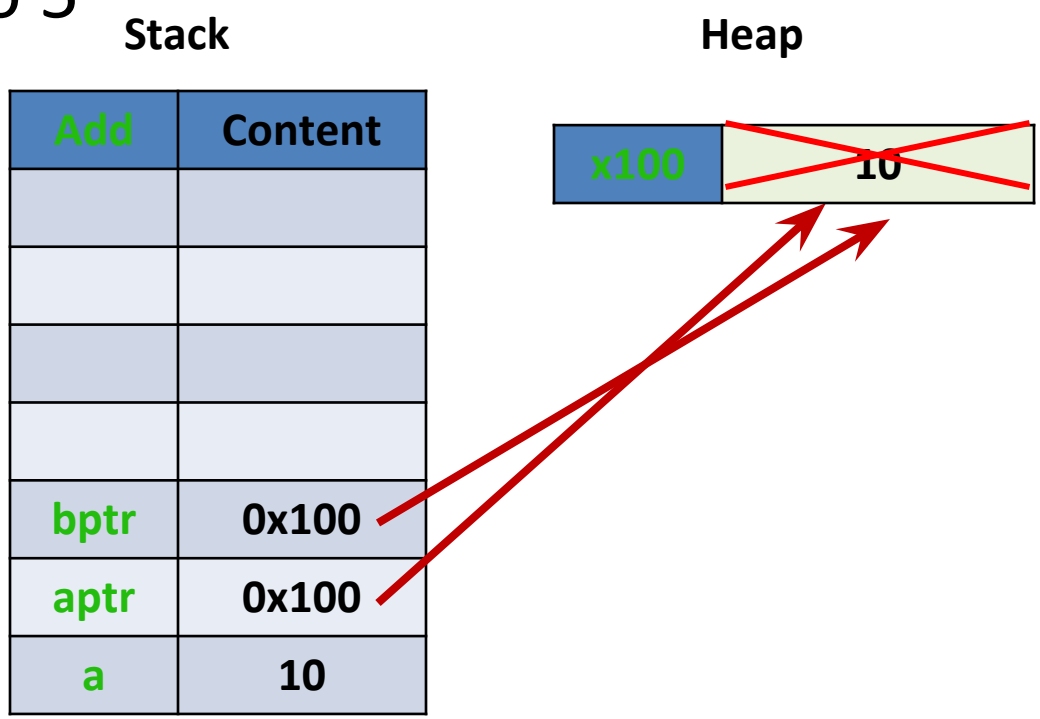
Int* bptr = aptr;

Delete aptr;

***bptr = a*5;**

Cout<<*aptr

<<*bptr<<endl;



Bptr is Dangling Pointer. Trying to access illegal memory.

Class Activity 3 - Answer

Int a = 10;

Int* aptr = new int(5); //allocates one integer on heap and initializes it to 5

*aptr = a;

Int* bptr = aptr;

Delete aptr;

***bptr = a*5; //Error: bptr is dangling pointer, trying to access illegal memory**

Cout<<*aptr<<*bptr<<endl;

Class Activity 4

```
Int a=999;
Int* aptr = new int; //taking one integer from
//heap
aptr = &a;
Cout<<*aptr<<endl;
```

Class Activity 4

Int a=999;

Int* aptr = new int;

aptr = &a;

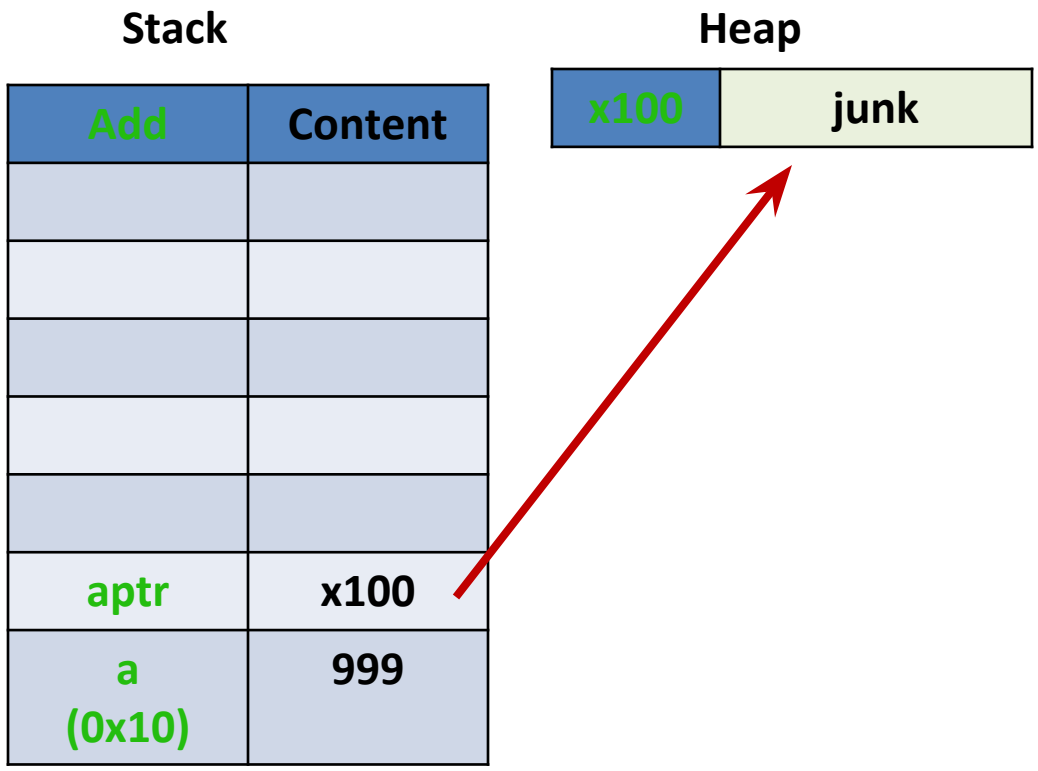
Cout<<*aptr<<endl;

Stack

Add	Content
a (0x10)	999

Class Activity 4

```
Int a=999;
Int* aptr = new int;
aptr = &a;
Cout<<*aptr<<endl;
```

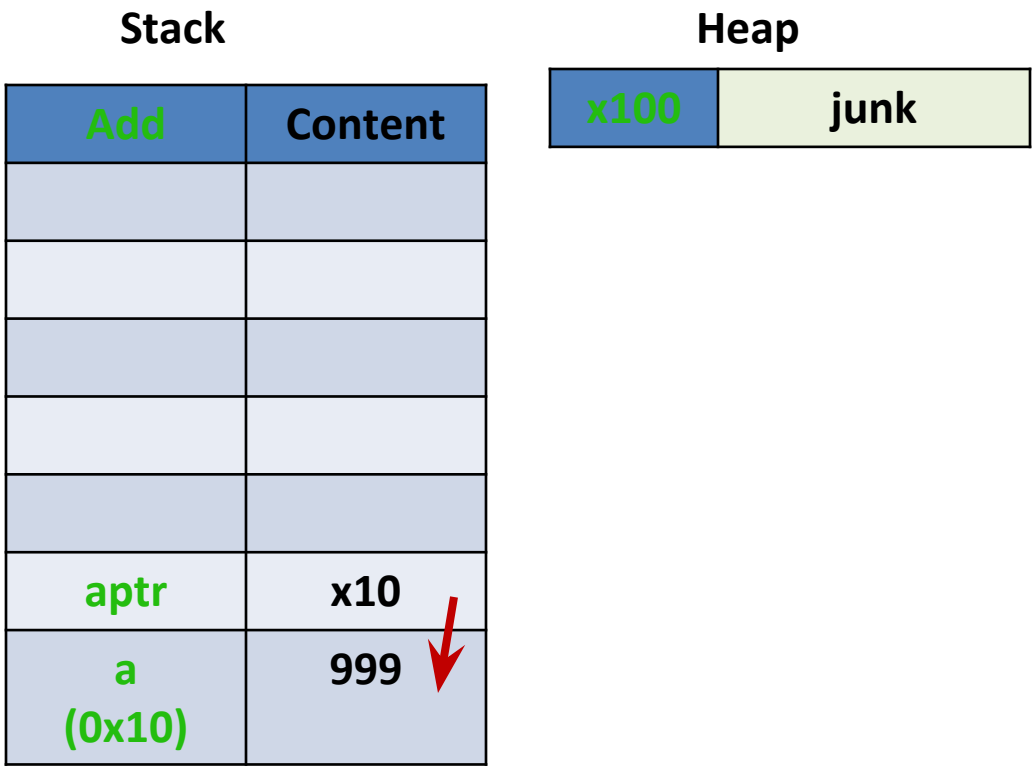


One integer allocated on heap.

Class Activity 4

```
Int a=999;
Int* aptr = new int;
aptr = &a;
Cout<<*aptr<<endl;
```

Memory leaked.



Class Activity 4

Int a=999;

Int* aptr = new int; //taking one integer from //heap

aptr = &a;//Logical Error: Memory Leakage.(This is not exception so next line will print value 999 and program will terminate successfully)

Cout<<*aptr<<endl;

Class Activity 5

```
Int* aptr = new int; //int* aptr = 0;
Int size = 10;
Cin>> size;
Aptr = new int[size];
```

Class Activity 5

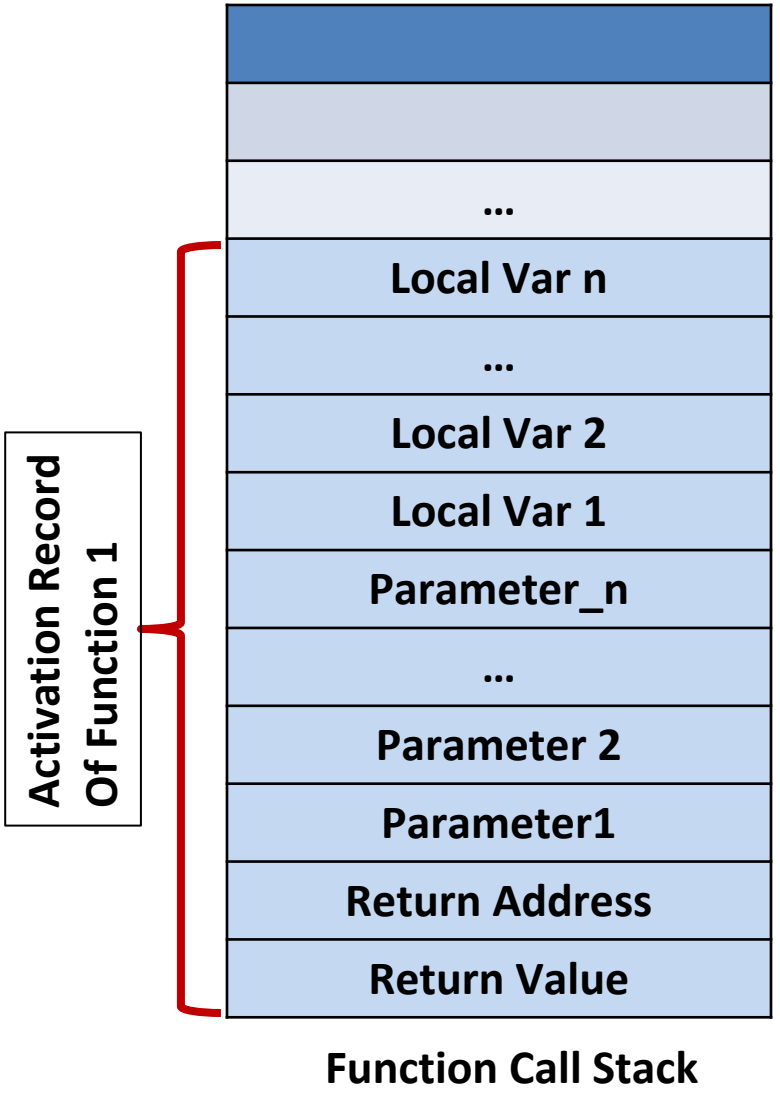
```
Int* aptr = new int;
Int size = 10;
Aptr = new int[size]; //Previous Memory Leaked
```

Functions

- Why do we use Functions?
  - Divide and Conquer Approach
  - Software Reusability
  - Avoid Repetition of Code
  - Dividing a program into meaningful functions makes it easier to debug and maintain

Function Calls

- Function Call Stack
- Activation Record/Stack Frame
- Stack Overflow
- Function Call Overhead
  - Creation of A.R.
  - Pop from Stack
- Inline Functions



Function Overloading

```
Int Sum(int a, int b)
{
}
Int Sum (int a, int b, int c)
{
}
```

- Used when purpose of function is same but signature or steps are different.

Function Calls (contd.)

```
void Fun1(int x, int& y)
{
cout<<"\n\nEntered
Fun1().\n";//0x1111
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

x = 111;
y = 222;

cout<<"\n\nLeaving Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

cout<<"\n\n&x
=\t"<<&x<<"\t&y=\t"<<&y<<endl;
}
```

```
void TestParameters()
{
int a = 10;//0x2222
int b = 20;

cout<<"Before calling Fun1().\n";
cout<<"a =\t"<<a<<"\tb =\t"<<b<<endl;

Fun1(a,b);

cout<<"\n\nAfter Fun1().\n"; //Line Address xCDFE
cout<<"a =\t"<<a<<"\tb =\t"<<b<<endl;

cout<<"\n\n&a =\t"<<&a<<"\t&b =\t"<<&b<<endl;
}
void main()
{
TestParameters();
} //0xBDF
```

Function Calls (contd.)

```
void Fun1(int x, int& y)
{
int z = 999;

cout<<"\n\nEntered Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

x = 111;
y = 222;

cout<<"\n\nLeaving Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

cout<<"\n\n&x
=\t"<<&x<<"\t&y=\t"<<&y<<endl;
}
```

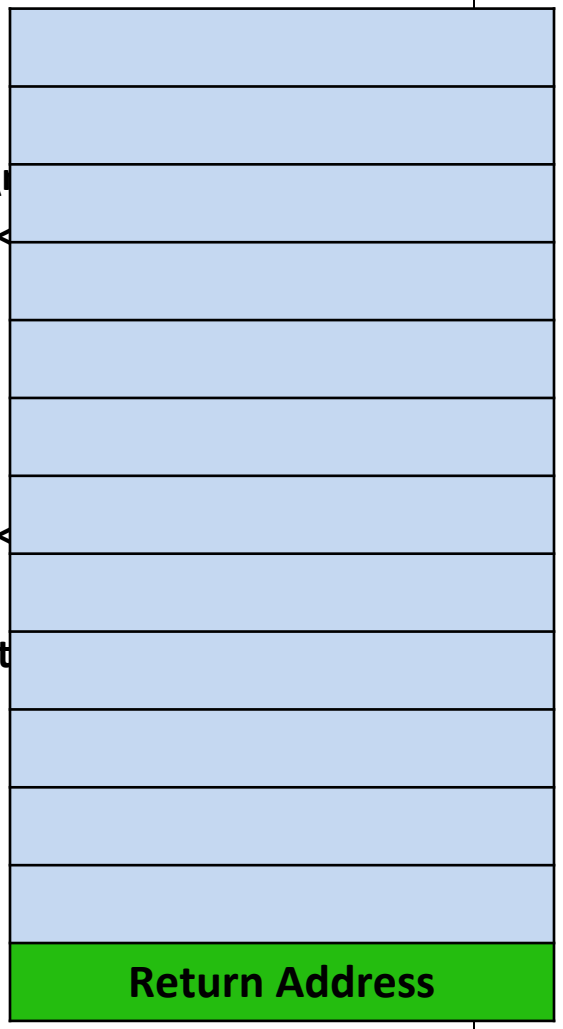
```
void TestParameters()
{
int a = 10;
int b = 20;

cout<<"Before calling Fun1().\
cout<<"a =\t"<<a<<"\tb =\t"<<

Fun1(a,b);

cout<<"\n\nAfter Fun1().\n";
cout<<"a =\t"<<a<<"\tb =\t"<<

cout<<"\n\n&a =\t"<<&a<<"\t
}
void main()
{
TestParameters();
}
```



Function Call Stack

Function Calls (contd.)

```
void Fun1(int x, int& y)
{
int z = 999;

cout<<"\n\nEntered Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

x = 111;
y = 222;

cout<<"\n\nLeaving Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

cout<<"\n\n&x
=\t"<<&x<<"\t&y=\t"<<&y<<endl;
}
```

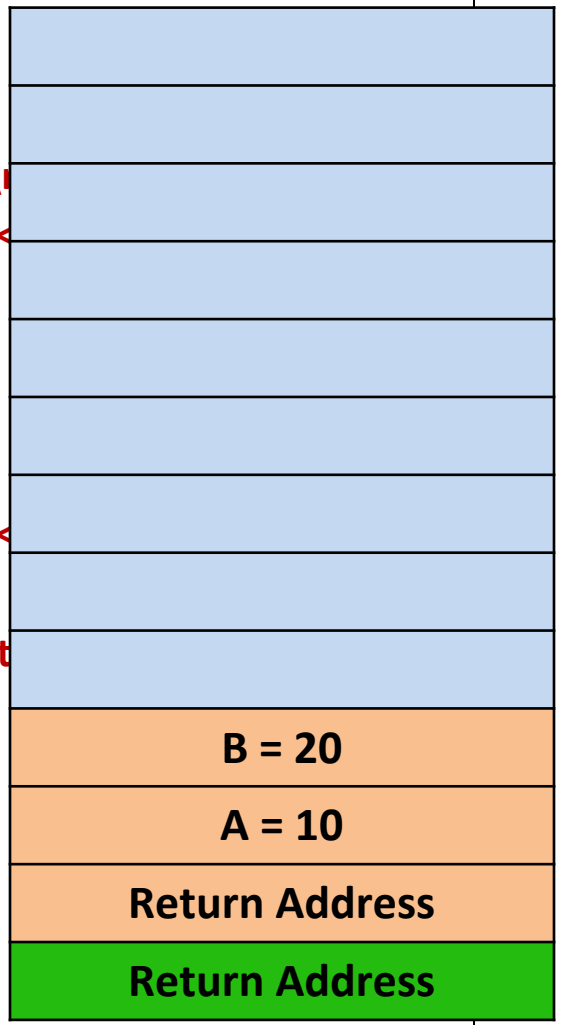
```
void TestParameters()
{
int a = 10;
int b = 20;

cout<<"Before calling Fun1().\
cout<<"a =\t"<<a<<"\tb =\t"<<

Fun1(a,b);

cout<<"\n\nAfter Fun1().\n";
cout<<"a =\t"<<a<<"\tb =\t"<<

cout<<"\n\n&a =\t"<<&a<<"\t
}
void main()
{
TestParameters();
}
```



Function Calls (contd.)

```
void Fun1(int x, int& y)
{
int z = 999;

cout<<"\n\nEntered Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

x = 111;
y = 222;

cout<<"\n\nLeaving Fun1().\n";
cout<<"x =\t"<<x<<"\ty =\t"<<y<<endl;

cout<<"\n\n&x
=\t"<<&x<<"\t&y=\t"<<&y<<endl;
}
```

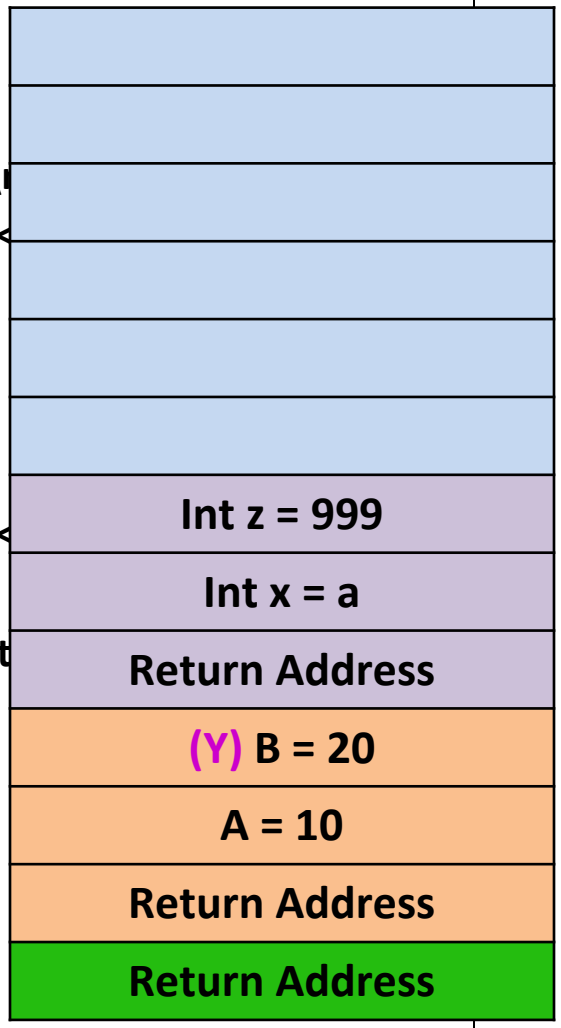
```
void TestParameters()
{
int a = 10;
int b = 20;

cout<<"Before calling Fun1().\
cout<<"a =\t"<<a<<"\tb =\t"<<

Fun1(a,b);

cout<<"\n\nAfter Fun1().\n";
cout<<"a =\t"<<a<<"\tb =\t"<<

cout<<"\n\n&a =\t"<<&a<<"\t
}
void main()
{
TestParameters();
}
```



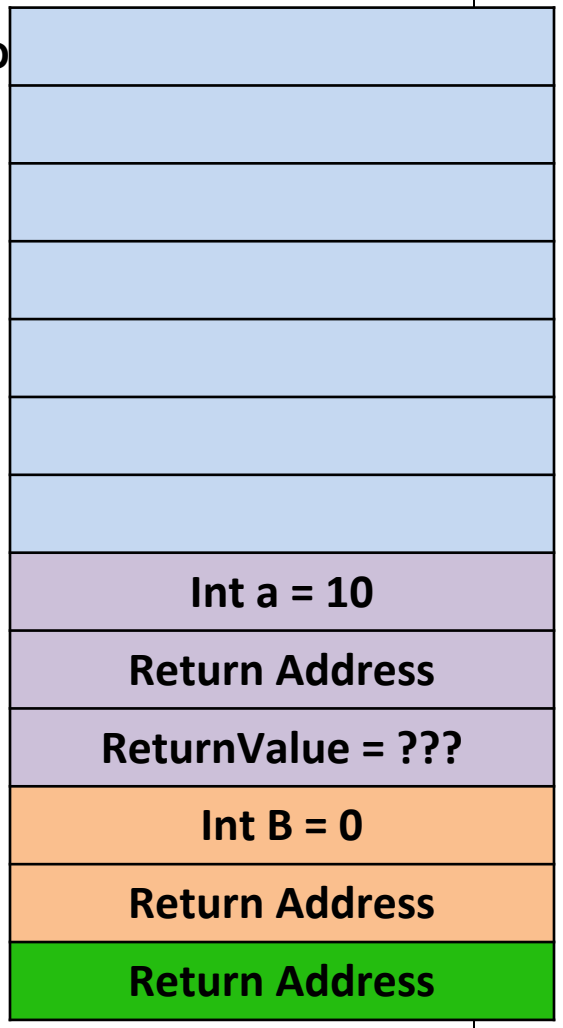
Function Calls – Return Values

```
int ReturnSomething()
{
int a = 10;
return a;
}
```

```
void TestReturnValue()
{
int b = 0;
b = ReturnSomething();//0xBD

cout<<"b = "<<b<<endl;
}

void main()
{
//TestParameters();
TestReturnValue();
}//0xABC
```



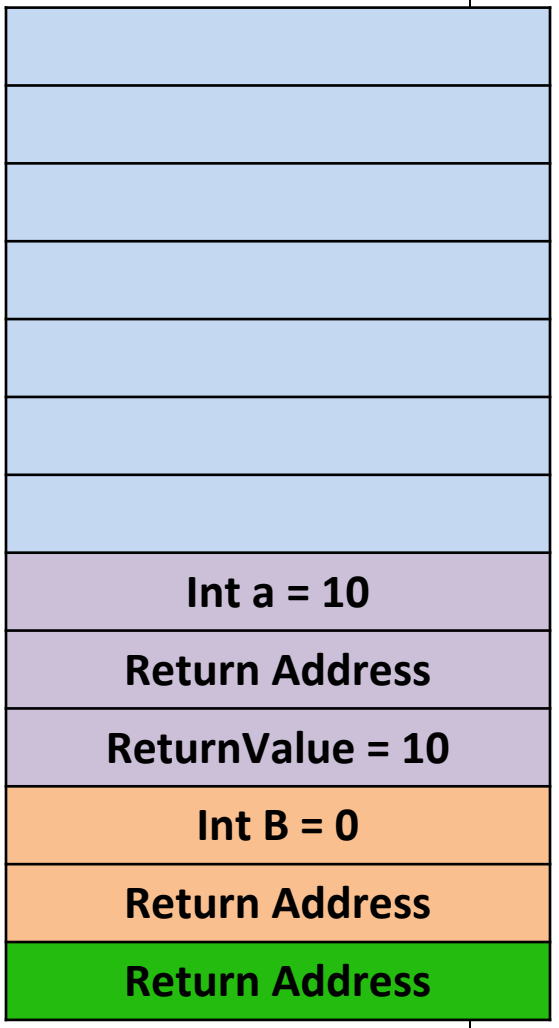
Function Calls – Return Values

```
int ReturnSomething()
{
int a = 10;
return a;
}
```

```
void TestReturnValue()
{
int b = 0;
b = ReturnSomething();

cout<<"b = "<<b<<endl;
}

void main()
{
//TestParameters();
TestReturnValue();
}
```



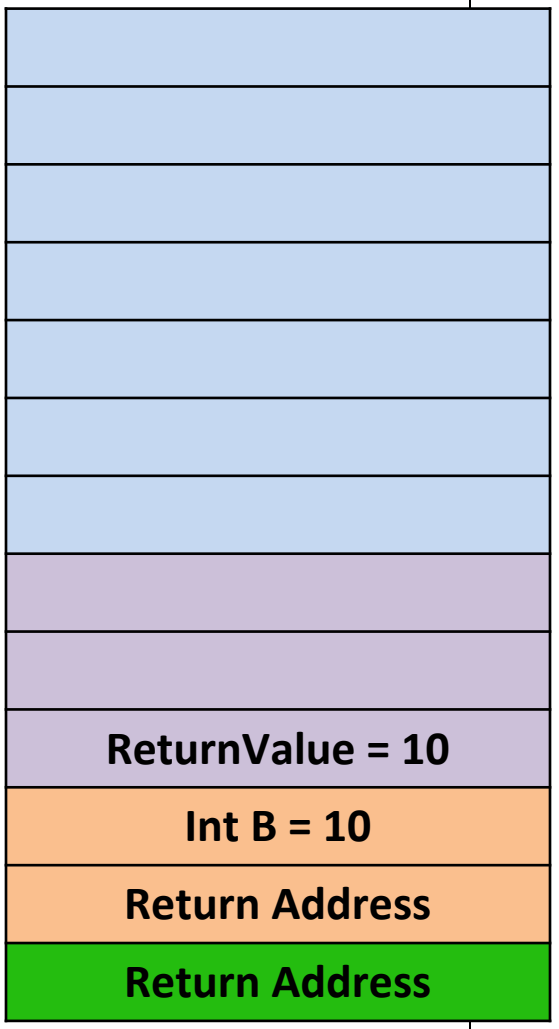
Function Calls – Return Values

```
int ReturnSomething()
{
int a = 10;
return a;
}
```

```
void TestReturnValue()
{
int b = 0;
b = ReturnSomething();

cout<<"b = "<<b<<endl;
}

void main()
{
//TestParameters();
TestReturnValue();
}
```



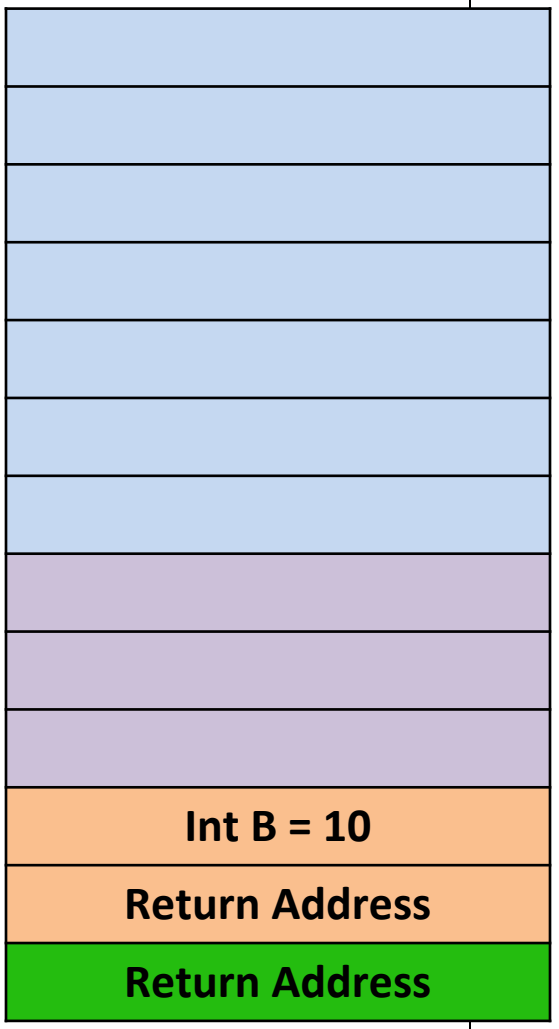
Function Calls – Return Values

```
int ReturnSomething()
{
int a = 10;
return a;
}
```

```
void TestReturnValue()
{
int b = 0;
b = ReturnSomething();

cout<<"b = "<<b<<endl;
}

void main()
{
//TestParameters();
TestReturnValue();
}
```



Functions

- Pass By Value VS Pass By Reference
  - Pros and Cons???