Lecture 12.1 CSI333 Storage Areas II

```c
#include <iostream.h>
int * create123( void )
{
    int A[3]; //DEFINE an automatic array
    return A; // Don’t do this! (return a
    // dangling pointer)
}
int main(int argc, char *argv[])
{
    int *pint;
    pint = create123();
    return 0;
}
```
$ g++ -o slide1 slide1.cpp
slide1.cpp: In function `int * create123()’:
slide1.cpp:5: warning: address of local variable ‘A’ returned
$ slide1
123
1073783640-1073743192134514440
$

int * create123( void )
{
  int A[3];  //DEFINE an automatic array
  return A;  // Don’t return a dangling pointer!
}

A[i]  One array element. It’s an int variable
A    Array name. It denotes the constant equal to the address of element 0.

A[0] = 0; makes sense.    A = 0; is illegal.

&C[0]  same meaning as   A

C/C++ rules:
A[0]  same meaning as   *A
&A[i]  same meaning as   A+i
A[i]  same meaning as   *(A+i)

Advanced C++: Programmer defined data types can have their own operator[].
int * create123( void )
{
    int A[3]; //DEFINE an AUTOMATIC variable
    A[0] = 99; // (A is a constant pointer)
    return A; // VERY BAD THING TO DO!
}

int * create123( void )
{
    int * A = new int[3]; //DEFINE a FREE-STORE variable
    A[0] = 99; // A is NOT an array!!
    return A; // OK. A is a pointer variable.
    // Caller should run delete to reclaim space.
}

int * create123( void )
{
    static int A[3]; //DEFINE a STATIC variable
    A[0] = 99;
    return A; // OK.
    // the array exists forever..}

int * create123( void )
{
    int A[3]; //DEFINE an AUTOMATIC variable
    A[0] = 99; // (A is a constant pointer)
    return A; // VERY BAD THING TO DO!
    // The compiler DID warn you!
}

create123:
    sub $sp, $sp, 12 #get stack space for array
    li $t0, 99
    sw $t0, 0($sp) #store 99 in A[0]
    move $v0, $sp #$v0 will = DANGLING POINTER!!
    add $sp, $sp, 12 #array’s space is GARBAGE
    jr $ra #$v0=dangling pointer value:(
int * create123( void )
{
    int * A = new int[3];  //DEFINE a FREE-STORE variable
    A[0] = 99;  // A is >>NOT<< an array!!
    return A;  // OK.  A is a pointer variable.
    //Caller should run delete to reclaim space.  
}

create123:
    subi  $sp,$sp,8  #get stack space for pointer A and $ra
    sw  $ra, 4($sp)  #save $ra: create123 WILL call fun's
    li  $a0, 12    #to request 3*4 bytes from malloc
    jal malloc      #memory alloc. library function
    sw  $v0, 0($sp) #A=$v0= pointer to malloc'd space
    li  $t0, 99
    sw  $t0, 0($v0) #store 99 in first alloc'd word
    lw  $ra, 4($sp) #restore $ra.
    addi $sp, $sp, 8  #pop off stack frame
    jr  $ra  #$v0 = addr returned by malloc

int * create123( void )
{
    static int A[3];  //DEFINE a STATIC variable
    A[0] = 99;
    return A;  // OK.
    // the array exists forever..}
    .data
    A: .word  0, 0, 0  #assemble 3 word array
    .text
    create123:
        la  $v0, A  #get address of array elt. 0
        li  $t0, 99
        sw  $t0, 0($v0) #store 99 in A[0]
        jr  $ra  #$v0=addr. symbolized by A
Stroustrup (4.9.6): “something in memory ... is the simplest and most fundamental notion of an object.
... an object" is a contiguous region of storage.

(10.4.3): Lifetimes, extents, ways of creation/destruction:

**Named automatic object**(10.4.4):
Created each time its declaration is encountered and destroyed each time the program exits the block in which it occurs.

**Free-store object**(10.4.5):
Created using new and destroyed using delete.

**Local static object**(10.4.8):
Created the first time declaration is encountered in execution and destroyed once at the termination of the program.

**Global static object**(10.4.9): Created once “at the start of the program” and destroyed once at termination.

*(not an “object oriented programming” class object)*

---

**Memory areas used for objects**

Named automatic object: System Stack frame for THE activation that created this object.

Free-store object: Heap: within data structures managed by memory allocation library functions.

Local/Global static object: .data segment.

**What Destruction Means**

1. RECYCLE the object’s memory space (MAKE IT GARBAGE!!)
2. run any “destructor function” first (may be a csi333 topic later..))