Pointers Cheat Sheet

Albert Hsu

December 16, 2014

1 Basics

Code	Result
char *ptr;	ptr is set to NULL. No memory allocated.
char *ptr, ptr2;	ptr2 is not a pointer but ptr is pointer.
char **ptr;	A double pointer is set to NULL. No memory allocated.
char array[10];	A array size 10 on stack.
char *a[20];	An array of char pointer of size 20.
char a[1][2];	A 2-D array on stack.
char a[x];	A array of size x. x has to be const.
char *a = new char[x];	A dynamic array of size x on heap. x can change during runtime.
char **a = new char*[x];	A Dynamic array of size x of char pointer on heap. x can change during runtime.
char **a = new char[x][y];	A 2-D Dynamic array on heap. x and y can change during runtime.

Code	Is it a memory address	Or content of the address
char* ptr;	yes	no
ptr;	yes	no
*ptr;	yes	no
&variable;	yes	no
array;	yes	no
*array;	no	yes
array++;	yes	no
*(array++);	no	yes
array[1];	no	yes
doublearray[0][0];	no	yes
**doublearray;	no	yes
doublearray;	yes(pointing to another pointer)	no
doublearray+3;	yes(pointing to another pointer)	no
*doublearray;	yes	no
doublearray[0];	yes	no
*doublearray[0];	no	yes

Note that null pointers doesn't have the apporiate size to store it's data type. To solve this, allow the pointer to point to memory space that are already allocated for that data type. You can point to an exsisting variable of same data type or use new operator or malloc function.

Remeber a memory address cannot be set to a data type, it can only contain memory address.

Memory Allocation

operator/function	What it does
new(c++)	Allocate the right amount of memory on heap. It returns a pointer.
new[](c++)	Allocate array of the right amount of memory on heap.
delete(c++)	Free the allocated memory on heap that were allocated by new.
delete [](c++)	Free the array of allocated memory on heap that were allocated by new[].
malloc(size_t size)	Allocated size amount of bytes of memory. Returns a pointer pointing to it.
realloc(void* ptr, size_t size)	Re-allocate the pointer to size amount of bytes of memory.
calloc(size_t num, size_t size)	Allocated block of memory for an array of size num and each of them size bytes long
free(void* ptr)	De-allocated memory pointed by ptr.

Syntax	Technical Term	Meaning
*	Dereference	-Content of object a pointer is pointing to. -To declare a pointer
&	Unary	Address of a variable. Referencing a variable
**	Double pointer	Pointer pointing to another pointer
->	Structure derefence	This combines dereference and dot operator.
[]	Array subscript	For E1[E2], the array subscript is exactly same as *(E1+E2). You can see the order doesn't matter so 2[array] is correct since the complier will see it as *(2+array).

2 Function Pointers

Assume that two function both return nothing and takes in one int called func and func2. Remember that function pointer's type and parameters have to be the same just like a char pointer must point at a char type.

Assume that func and func2 are already declared that return nothing and takes

Code	Result
void (*f_ptr)(int);	Declared a function pointer that takes in one int and return nothing.
f_ptr = func;	f_ptr is now pointing to func.
(*f_ptr)(2);	Calls func and pass in 2.
f_ptr(2);	No different from (*f_ptr)(2);
(**f_ptr)(2);	No different from (*f_ptr)(2);
f_ptr = func2;	f_ptr is now pointing to func2.
f_ptr = &func2;	No different from f_ptr = func2;
f_ptr = *func2	No different from f_ptr= func2;
f_ptr = **func2	No different from f_ptr= fun2;

no parameters. Here are some code that shows function using function pointers

Code	Result
void f(void(*a)());	Function prototype for a function that return nothing and takes in one function pointer.
int f2(void(*b)(), int c);	Function prototype for a function that return a int and takes in one function pointer and a char.
void (*f_ptr)() = func;	Declare a void function pointer and point it to func.
void (*f_ptr2)() = func2;	Declare a char function pointer and point it to func2.
f(f_ptr);	Calls f and pass in func.
f2(f_ptr2, 5);	Calls f2 and pass in func2 and 5.
f(f_ptr2);	Calls f and pass in func2.
f_ptr = f_ptr2;	f_ptr is now pointing to func2.
f2(f_ptr, 5);	Calls f2 and pass in func2 and 5.

3 Edge Cases/Errors

Bad Code	Error	Explanation
array[array.size()];	Out of bound. Could cause a segmentation fault.	It's trying to access memory not allocated for that array. Always check your array bounds.
int *ptr; cout <<*ptr <<endl;	Segmentation fault.	You can't use NULL pointers. Always check if pointers are NULL pointers before using.
int *ptr = 100;	Compiler error	100 is not a int pointer type. Cannot assign values to pointers.
cout <<ptr <<endl;	Runtime	It will print out the actually address. Usually not what you want.
if(ptr == ptr2)	Runtime	Compiler will compare the address of ptr and ptr2. Dereference them first.
char a[];	Compiler error	Compiler need to know the size of the array. Use memory allocation operator or function.
char *a[];	Compiler error	Same as above.
char a[][];	Compiler error	Same as above.
char **a = new char;	Compiler error	Double pointer cannot point to a int. Must point to another pointer.

int *ptr, ptr2;
ptr1 = new int;
ptr2 = ptr1;
delete p1;
*ptr2 = 2; //Error
	Segmentation fault.
	Dangling pointer
		ptr1 and ptr2 points to the same memory block so when you delete p1, the memory is no longer allocated. When trying to access ptr2, you're trying to access non allocated memory.

int *ptr; ptr = new int; ptr = new int;	Memory Leak.	The memory allocated for the first new operator is no longer accessible
void function() { int *ptr = new int; }	Memory Leak.	Not freeing the allocated memory. Since this is inside a function, each function call will create inaccessible memory on heap.