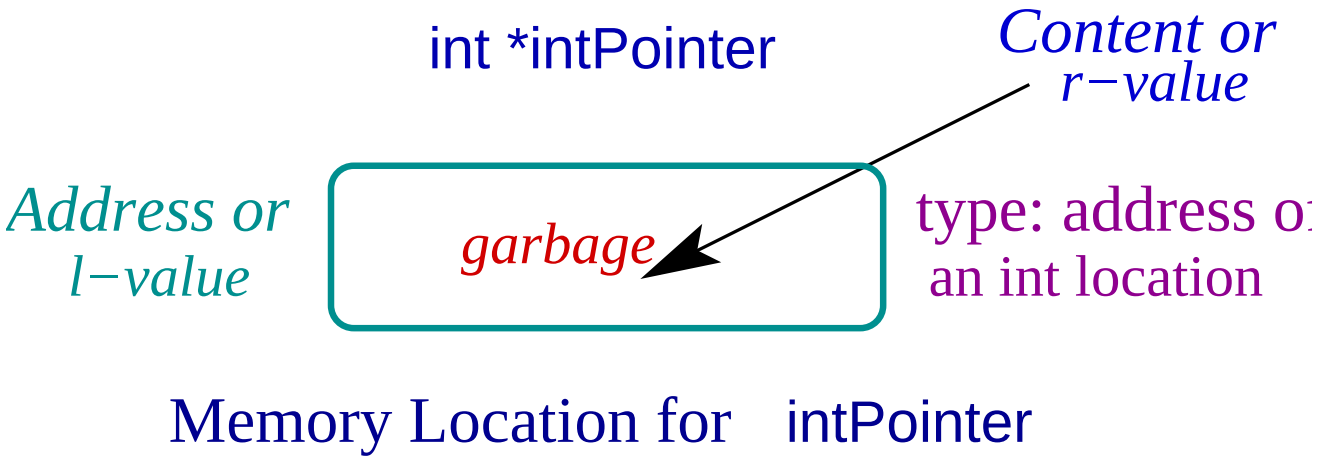
Pointer Variable, Memory Location & Content

```
int *intPointer
```

- Address of a variable can be extracted by the unary operator '&'.
- Address of a loaction is a storable value.
- A variable of type `int *` can store the address of a location of type `int`.


int *intPointer
Content or
r−value
Address or
l−value
garbage
type: address o
an int location
Memory Location for intPointer

- Memory is allocated to a variable of type 'int *' (pointer to an int) like any other variable. Its size depends on the machine architecture[a].
- Pointer location does not contain any valid address unless it is initialized.

[a]The size is 8-bytes on a x86_64 machine.

sizeof

- The unary operator sizeof can be used to find the size of a type or of a variable.
- The size of pointer variable of all types are identical. Then, a natural question is, why does C language uses different type names for them.

```
#include <stdio.h>
int main() // temp11.c
{
    char n, *p ;
    printf("sizeof n: %ld\n", sizeof n) ;
    printf("sizeof p: %ld\n", sizeof p) ;
    printf("sizeof(char): %ld\n",sizeof(char)) ;
    printf("sizeof(char *): %ld\n",
                    sizeof(char *)) ;
    return 0 ;
}
```

Output

```
$ cc -Wall temp11.c
$ ./a.out
sizeof n:  1
sizeof p:  8
sizeof(char):  1
sizeof(char *):  8
```

```
#include <stdio.h>
int main() // temp12.c
{
    printf("sizeof(char *): %ld\n",
                    sizeof(char *)) ;
    printf("sizeof(int *): %ld\n",
                    sizeof(int *)) ;
    printf("sizeof(float *): %ld\n",
                    sizeof(float *)) ;
    return 0 ;
}
```

Output

```
$ cc -Wall temp12.c
$ ./a.out
sizeof(char *):  8
sizeof(int *):  8
sizeof(float *):  8
```

(mis)Use of a Pointer

The unary operator '$*$' (not to be confused with the binary multiplication operator) applied to an address or pointer to any location of any type[a] gives the object bound to that location.

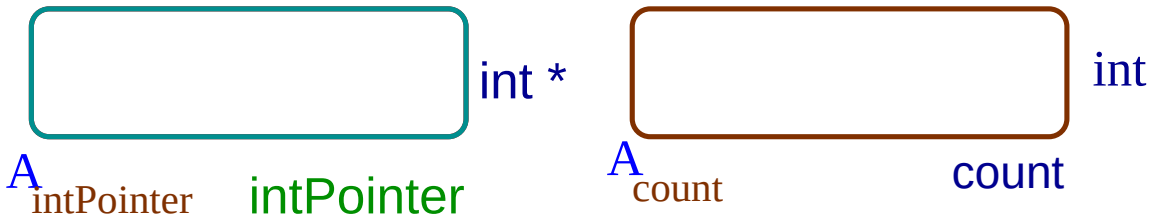
[a] It is polymorphic.

```
#include <stdio.h>
int main() // temp7.c
{
    int count = 10, *intPointer = &count;
    printf("count: %d, *intPointer: %d\n",
                count, *intPointer);
    count = count + 5 ;
    *intPointer = *intPointer*10 ;
    printf("count: %d\n", *intPointer);
    return 0 ;
}
```

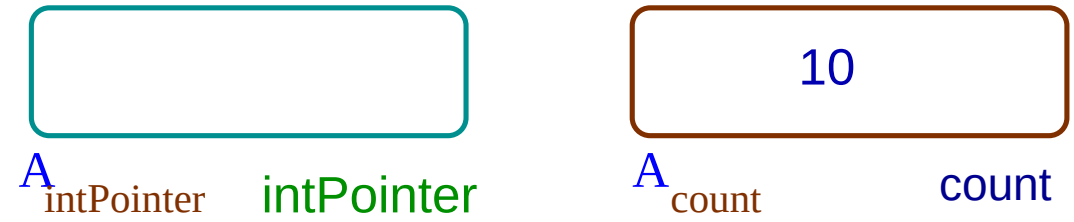
Output

```
$ cc -Wall temp7.c
$ ./a.out
count:  10, *intPointer:  10
count:  150
```

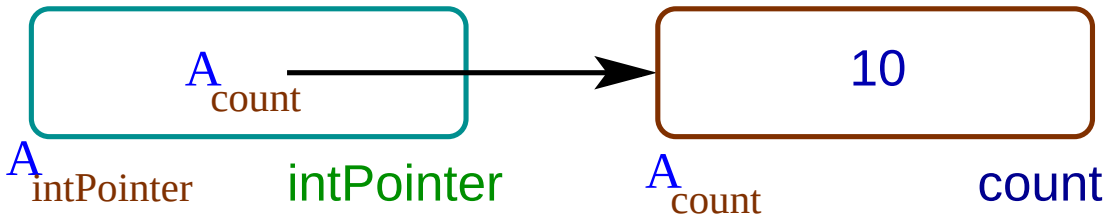
int *intPointer, count ;



count = 10 ;

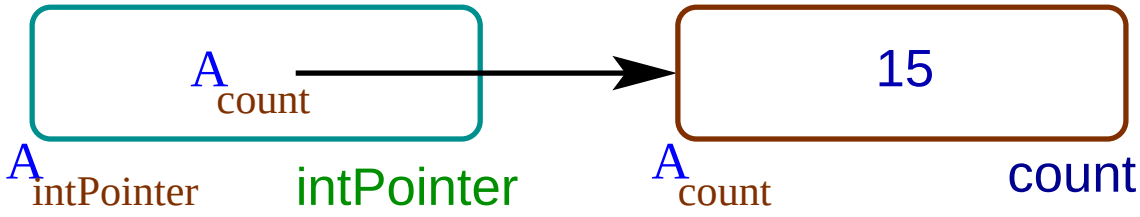


intPointer = & count ;

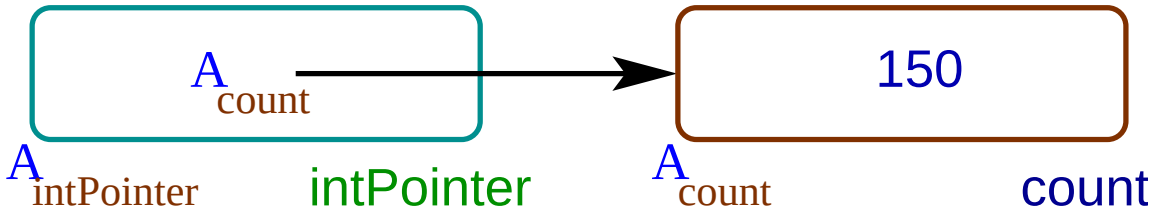


- The `int` variable `count` is initialized to 10.
- The `int *` variable `intPointer` is initialized with the address of the location of `count`.

count = count + 5 ;



*intPointer = *intPointer*10 ;



- The variable `intPointer` stores the address of the object `count`.
- The expression `*intPointer` is equivalent to the object `count`.
- If the '*' operator is applied to an illegal address (pointer), there will be an error (a segmentation fault).

```
#include <stdio.h>
int main() // temp13.c
{
    int *p = (int *)100 ; // illegal
    printf("*p: %d\n", *p) ;
    return 0 ;
}
```

Output

```
$ cc -Wall temp13.c
$ ./a.out
Segmentation fault
```