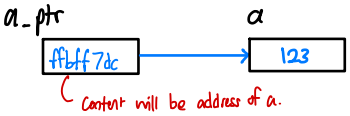
Pointers & Functions

C: - allows direct manipulation of memory contents using pointer
- have a set of bit manipulation operators, allowing efficient bitwise manipulation.

Pointers: - all variables have an address in memory
- usually system keeps track of the variable's address
- & address operator. %p for pointers.
- addresses are printed out in hexadecimal
- addresses varies from run to run.
- Pointer Variable (Pointer)



· declaration:

type *pointer_name;

"pointer_name is a pointer to variable of type type".

· assign value:

```
int a = 123; // need to declare first
int *a_ptr; // contains random data.
a_ptr = &a;
```

· pointers have their own address.

· Cannot access original variable through the pointer

· Use indirection operator/dereferencing.

*a_ptr Compiler see address of a, goto address of a, see value, then return value.

Can also write and read the value

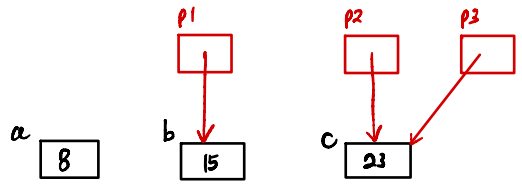
*a_ptr = 456 ≡ a = 456 → *a_ptr is synonymous with a.

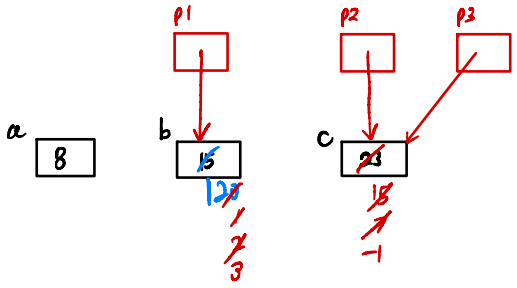
Example

```
int a = 8, b = 15, c = 23;
int *p1, *p2, *p3;
p1 = &b;
p2 = &c;
p3 = p2;
printf("1: %d %d %d\n", *p1, *p2, *p3);   → 1: 15 23 23
*p1 *= a;   → *p1 = *p1 * a. = 120
            (*p1 = 15, a = 8)
while (*p2 > 0) {
    *p2 -= a;
    (*p1)++;        *p1++ ie. *(p1++)
}
```

deref has lower precedence than ++ ⇒ pointer incre first then deref.

```
printf("2: %d %d %d\n", *p1, *p2, *p3);   → 2: 123 -1 -1.
printf("3: %d %d %d\n", a, b, c);         → 3: 8 123 -1
```





Pointer Arithmetic

- p++
- int takes in (4 bytes but dependent on Machine Architecture)
- int*++ · ffbff0a4 —(+4, Size of one int)→ ffbff0a8
- float*++ · ffbff0a0 —(+4, Size of one float)→ ffbff0a4
- char*++ · ffbff09f —(+1, Size of one char)→ ffbff0a0
- double*++ · ffbff090 —(+8, Size of one double)→ ffbff098.

Address will increase by the size of the data type it is pointing to.

Segmentation fault

eg. `int *n;` // no initialization ∴ random memory address (might belong to another program).

`*n = 123` // attempts to change the value in random memory address

↓

Segmentation fault by OS ⇒ often result in core dumped (remove the file 'core')
⇒ contains all memory allocated to you.

→ When one program tried to mess with the memory of another program.

Why pointers?

- pass address of a variable into a function.
  - ⇒ want the function to modify the value of a variable.
  - ⇒ function can then dereference the variable and change the value.
- pass arrays into the function
  - ⇒ can pass address of the first element of the array.
  - ⇒ use pointer arithmetic to access individual elements of the array.

Functions

- function prototype – args., return values. eg. double pow(double x, double y) ← return type.
- <stdio.h>, <math.h> w/ gcc -lm.
- scanf("%d %d", &x, &y)
  - only way for a function to modify x and y is with the address of x and y.
- user-defined functions:
  - if something is computed more than once.

```
double circle_area(double diameter){
    return PI * pow(diameter/2, 2);
}
```

- declare function prototype on top:

eg.

```
double circle_area(double);   ← func prototype: inform compiler of functions program might use.
                                 else implicit declaration warning & conflicting type (assumed return type int)
                                 ↑ do which 'type' code to generate.
int main(void){               ← start analyze program here!
    ⋮
}
double circle_area(double diameter){   ← func def.
    ⋮
}
```

Pass-by-value

- actual parameters are passed to the formal parameters by a mechanism known as pass-by-value

- actual parameters are COPIED into formal parameters.
  - actual parameters ↓ in the main
  - formal parameters ↓ in the function.

- The value is being passed

- Call stack.
  - activation record for the function is created
  - clears the local variable after function exits
  - automatic variables ↺.
    - vs
    - Static variable (won't disappear).

- void *ptr.
  - used to declare parameters in a function.
    can pass in any data type

How do we allow a function to return more than one value or modify values of variables defined outside it?

Function with Pointer Parameters

⇒ Use pointers!

eg.

```
void swap(int *, int *);
int main(void){
        :
    swap(&a, &b);
        :
}

void swap(int *ptr1, int *ptr2) {
    int temp;
    temp = *ptr1; *ptr1 = *ptr2; *ptr2 = temp;
}
```