Intro. Computing with the C Programming Language

Pointers

- PIC: Ch. 10. Pointers
- PIC: Ch. 14. Advanced topics

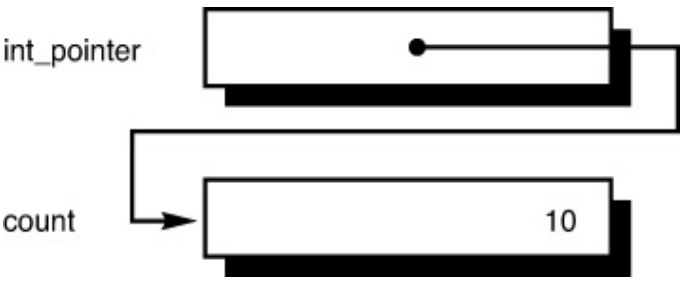
Shin Hong

23 November 2023

Pointer Variable

- A pointer is a number indicating the memory location (memory address) of a variable, by which the variable can be read or written indirectly
- A pointer variable is a variable to hold a pointer

```
int count = 10 ;
int * int_pointer ;

int_pointer = &count ;
```



- Examples: `pointer.c`, `char.c`

Pointer dereference operation

- A pointer dereference checks out the value at the memory location that the pointer indicates
  - often called as indirection operation

- Pointer dereference has higher precedence than arithmetic operations
  - e.g., `i2 = *p1 / 2 + 10;`

Pointer with Structure (1/2)

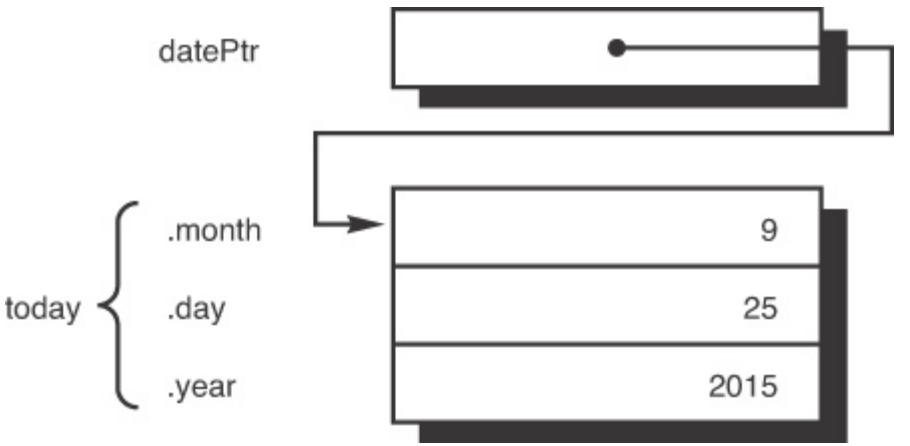
- The `struct` primitive can be used without the `typedef` primitive for defining a new structure
  - e.g.,

```
struct date {
   int month ;
   int day ;
   int year ;
} ;

typedef struct date date_t ;

struct date today ;
date_t yesterday ;
```

- We can define a pointer variable of a structure type as same for the primitive types
  - e.g., `struct date * p_date ;`

Pointer with Structure (2/2)

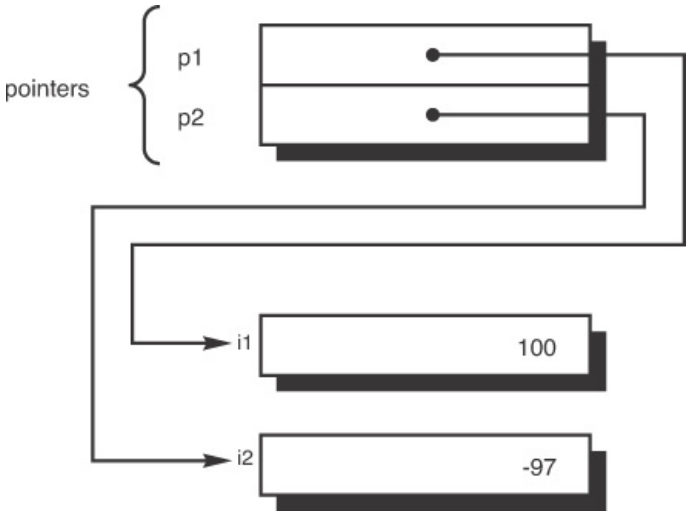
- A field of a structure variable can be referred by the arrow operator (e.g., `->`) with a pointer to the structure and the field name
  - e.g., `date.c`



Structures Containing Pointers

- A pointer can be a member of a structure
  - e.g.,

```
struct  intPtrs {
     int  *p1;
     int  *p2;
};
struct intPtrs  pointers;
```



Self-referential Structure

- A structure may have a pointer of its own
  - often, for representing relations in the same kind of objects
  - e.g.,

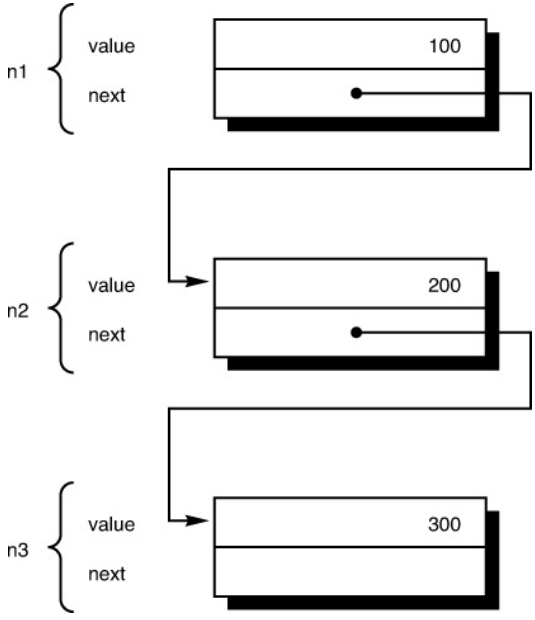
```
struct person {
   char name[32] ;
   struct person father ;
   struct person mother ;
} ;
```

```
struct person {
   char name[32] ;
   struct person * father ;
   struct person * mother ;
} ;
```

Linked List

- A linked list is a chain of structure variables with a value and a self-referential pointer
  - e.g., `linkedlist.c`

```
struct entry {
    int value ;
    struct entry * next ;
} ;
struct entry n1, n2, n3 ;
n1.value = 100 ;
n2.value = 200 ;
n3.value = 300 ;

n1.next = &n2 ;
 n2.next = &n3 ;
n3.next = NULL ;
```



Why Linked List is Needed

- To define/change an ordering among a set of data elements efficiently
- To increase/decrease the memory for storing data elements on demand (dynamically)
  - the capacity is fixed if an array is used

Dynamic Memory Allocation (Ch. 16)

- It is often desirable for a program to manage memory allocation directly
  - acquire more memory from the computer system
  - return the memory back to the system after the use
  - allocate a global variable of a certain size regardless of a function call

- `malloc`(*n*) allocates a *n*-byte memory at the heap region and returns its pointer

- `free`(*p*) releases the *n*-bytes memory from the pointer *p* which was allocated by a malloc

- e.g., `mem.c, readniv.c`

Linked List on Heap

- To store data elements of a type *K*, define a linked list node *L* which stores a *K* value and a pointer for type *K*.

- As a new element is given, an object of type *L* is newly allocated to store the given element and a new pointer to the next element

- e.g., `listheap.c`

String with dynamic memory allocation

- String duplication
- String concatenation

- Exercise.
  Write a function that receives a string, split the string by a whitespace then returns an array of the split strings

C Programming Contest

- Dec 8 Fri, 8-10 PM (120 min)
- Around 10 problems
- Closed everything (no internet, no personal data)
- Use replit.com
  - need to sign up if you do not have an account
  - put your first and last name to the account profile
  - https://replit.com/teams/join/whodntufxzxetztsmzbadhqtdqrfbzpn-HGUCContest