Introduction to C

Instructor: Vinicius Prado da Fonseca, PhD (vpradodafons@online.mun.ca)

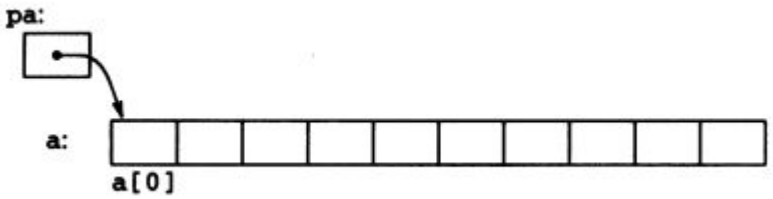
fileinput.c

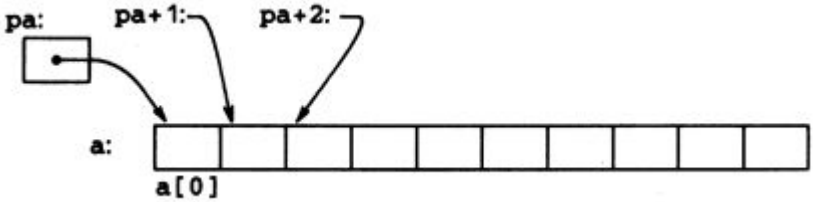
- File descriptor/pointer
- FILE* fd
- Cursor that walks the files each fscanf/fgets call

```
FILE* fd = fopen("points.csv", "rw");
fscanf(fd, "%s", mystring) != EOF
fscanf(fd, "%s", mystring) != EOF
fgets (mystring, 10, fd)!=NULL
```

5.3 Pointers and arrays

- Compilation time allocation (e.g. char line[MAXLINE])
  - Scope allocation/deallocation
- Static allocation, fixed size
  - Ex: Truncate smaller string with “\0”
- Request memory size during run time
- Dynamic allocation moving pointers

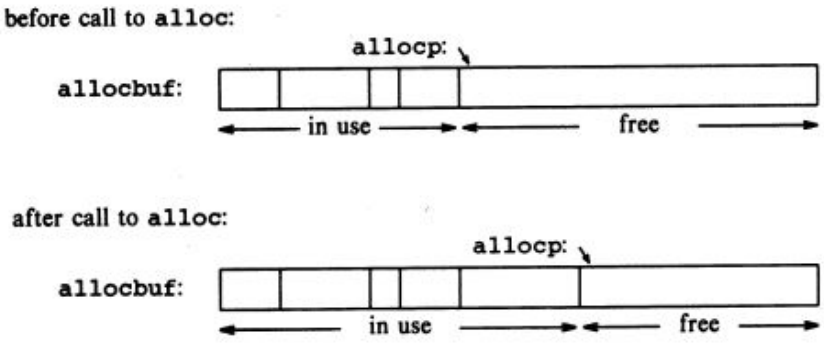




5.4 Address Arithmetic

Runtime allocation

- Stack-like allocation
- char allocbuff[ALLOCSIZE]
- char *allocp = allocbuf
- alloc(n)
  - Pointer p = allocp
    - allocp can be only visible by alloc and free
  - Move allocp requested size n
    - allocp + n positions
  - Return a pointer p
- afree(p)
  - Move the allocp to position p
  - Stack, last in first out.
- Correct call order



5.4 Address Arithmetic

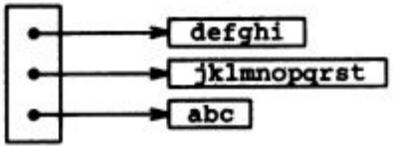
Runtime allocation

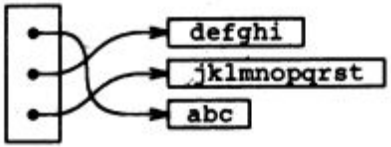
- allocbuff[ALLOCSIZE]
- alloc(n)
  - Pointer p = allocp
    - allocbuff and allocp can be only visible by alloc and free
  - Move allocp requested size n
    - allocp + n positions
  - Return a pointer p
- afree(p)
  - Move the allocp to position p
  - Stack, last in first out.
- Call alloc/afree correct order

```
char *alloc(int n) {
  // will it fit?
  if (allocbuf + ALLOCSIZE - allocp >= n) {
    allocp += n;
    return allocp - n;
  } else
    return 0;
}
```

5.6 Array of Pointers; Pointers to Pointers

- Pointers are variables themselves
  - Can be stored in arrays, like other variables
- Points to the first letter of the text
- Sorting is faster, just swap pointers
  - 5.6pointerarrays_a.c
- Complete sort example in the book





5.9 Pointers vs Multi-dimensional arrays

- Multi-dimensional arrays
  - int a[10][20];
  - 200 int-sized locations
- Pointers
  - int *b[10]
  - Using 20-element arrays
  - 200 ints + 10 pointers
- Advantage is rows may have different lengths

```
char aname[][15] = { "Illegal month", "Jan", "Feb", "Mar" };
```

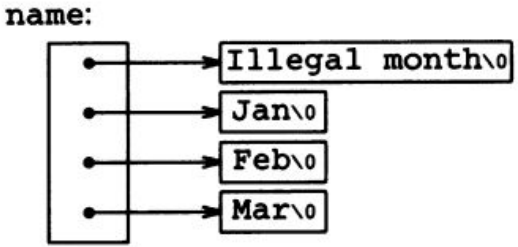
aname:

Illegal month\0	Jan\0	Feb\0	Mar\0
0	15	30	45

5.9 Pointers vs Multi-dimensional arrays

- Multi-dimensional arrays
  - int a[10][20];
  - 200 int-sized locations
- Pointers
  - int *b[10]
  - Using 20-element arrays
  - 200 ints + 10 pointers
- Advantage is rows may have different lengths

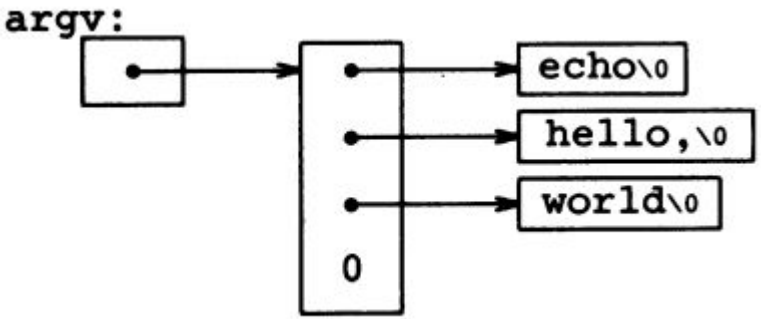
```
char *name[] = { "Illegal month", "Jan, "Feb", "Mar" };
```



5.10 Command-line Arguments

./echo hello, world

- argc
  - Argument counter
- argv
  - Argument vector



6.1 Basic of Structures

- Collection of one or more variables
  - Possibly different types
  - Grouped together
  - Treated as a unit
- Ex. Employee record
  - Name
  - Age
  - Job title
- typedef

```
struct point {
        int x;
        int y;
};
```

6.2 Structures and Functions

- Manipulate structures
- Pass components separately
- Pass the entire structure
- Pass a pointer

```
struct point init_pt(int x, int y) {
    struct point temp_pt;
    temp_pt.x = x;
    temp_pt.y = y;

    return temp_pt;
}
```

6.2 Structures and Functions

- Manipulate structures
- Pass components separately
- Pass the entire structure
- Pass a pointer

```
struct point init_pt(int x, int y) {
    struct point temp_pt;
    temp_pt.x = x;
    temp_pt.y = y;

    return temp_pt;
}
```

point-list.c

- Structure with a pointer to the next node
  Point.next = previous
- typedef
- pointnode is a tag
  - make the struct not unknown, reference after
- But typedef define PointNode as a type, like int. More common.

```
typedef struct pointnode {
        struct pointnode *nextpoint;
        int x;
        int y;
} PointNode;
```

point-list.c

- Dynamic allocation
  - malloc
- sizeof(PointNode)
  - Size in bytes
- (PointNode *)
  - Cast for the type we need

```
PointNode *palloc(void) {
        return (PointNode *) malloc(sizeof(PointNode));
}
```

point-list.c

- Printing example
- Follow the nextpoint
- Chained list

```
while (TRUE) {
        printf("%d, %d", iter->x, iter->y);
        iter = iter->nextpoint;
        if (iter == NULL) {
            printf("\n");
            break;
        }
        printf(" -> ");
    }
```

Try examples and code yourself

Really type the examples

Do not copy and paste

Repository