Introduction to C

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

Introduction

- Originally designed for and implemented on the UNIX OS DEC PDP-11 by Dennis Ritchie
- Not tied to any hardware or system
- Easy to compile and run in other systems
- General-purpose programming language
- Not strongly typed but strong type-checking
- Variety of data types (characters, integers and float point numbers)
- Derived data types created with pointers, arrays, structures and unions

Introduction

- Fundamental control-flow (if-else, switch, while, for, do, break...)
- Functions may return basic types, structures, unions or pointers
- Supports recursive function calls
- Preprocessing performs macro substitution
- Relatively low level, C deals with same objects that most computers would use (assembly)
- High level mechanisms need to be provided by explicitly-called functions
- Book: *C Programming Language (2nd Edition) by Brian W. Kernighan, Dennis M. Ritchie, 1988. (Pearson)*
- *https://github.com/vncprado/intro-to-c*

Environment

- Ubuntu

  ```
  $ sudo apt install build-essential
  ```

- Windows

  VSCode + Windows Subsystem for Linux (WSL), Brightspace instructions (Links and Resources)

- Mac

  Xcode, brew (package manager), gcc, clang (native)

What are we covering?

- A little of The Book: *C Programming Language (2nd Edition) by Brian W. Kernighan, Dennis M. Ritchie, 1988. (Pearson)*
  - Tutorial (ch1)
  - Functions and program structure (ch4)
  - Pointers (ch5)
  - Structures and unions (ch6)

1.1 Hello World!

- Environment test
  - Windows, Mac, Linux, VS Code
  - Text editor and shell (terminal)
  - # include <stdio.h>
  - int main()
- To compile

```
gcc hello_world.c
```

- To run

```
./a.out
```

hello.c

```c
#include <stdio.h>

int main () {
   printf("hello, world!\n");
}
```

```
vinicius@laptop:intro-to-c$ gcc hello.c
vinicius@laptop:intro-to-c$ ./a.out
hello, world!
```

Back to the basics

- Hello world
  - If else
  - for
  - Variables
  - printf

1.2, 1.3 Fahrenheits to Celsius

- C = (5/9) * (F - 32)
- 0, 20, 40, 60...300 F ---> C?
- While loop version
- Truncated integers
  - float - float point
  - char - character, single byte
  - short - short integer
  - long - long integer
  - double - double-precision float point

1.2Farhenheit-to-Celsius.c

```c
#include <stdio.h>

int main() {

   //int F, C;
   float F, C;
   int min_value = 0;
   int max_value = 300;
   int step = 20;

   F = min_value;
   while (F <= max_value) {
       C = (5.0/9.0)*(F-32.0);
       //printf("%d\t%d\n", F, C);
       printf("%8.2f\t%8.2f\n", F, C);
       F = F + step;
   }
}
```

1.2, 1.3 Fahrenheits to Celsius

- C = (5/9) * (F - 32)
- 0, 20, 40, 60...300 F ---> C?
- C and F are now floats
  - printf %f

1.2Farhenheit-to-Celsius.c

```c
#include <stdio.h>

int main() {

   //int F, C;
   float F, C;
   int min_value = 0;
   int max_value = 300;
   int step = 20;

   F = min_value;
   while (F <= max_value) {
       C = (5.0/9.0)*(F-32.0);
       //printf("%d\t%d\n", F, C);
       printf("%8.2f\t%8.2f\n", F, C);
       F = F + step;
   }
}
```

1.2, 1.3 Fahrenheits to Celsius

- For loop version
- Symbolic constants

1.3Celsius-to-Farhenheit-FOR.c

```c
#include <stdio.h>

#define name replacement_text
#define LOWER  0
#define UPPER  100
#define STEP   5

int main() {
 int celsius;

 printf("Celsius-Fahrenheit table\n");
 /* for (<initialisation>; <expression>; <expression>) */
 for (celsius = LOWER; celsius <= UPPER; celsius += STEP) {
   printf("%3.0f %6.1f\n", (float)celsius, (celsius * 9.0 / 5.0) + 32.0);
 }
}
```

1.5 Character input/output

- One character at a time
  - c = getchar()
  - putchar(c)
- ./a.out
  - type something
  - ctrl+d
- ./a.out < some_file

```
vinicius@laptop:book-examples$ ./a.out
test 123 test
test 123 test
```

1.5.1getchar-putchar.c

```c
#include <stdio.h>

int main() {
   int c;

   c = getchar();
   while (c != EOF) {
      //printf("%d\n", c);
      putchar(c);
      c = getchar();
   }
}
```

Input using scanf

- String example
- Scanf
  - Accept input from
    - String with format
  - Address of variable
  - Arrays variable name is a pointer

```
vinicius@laptop:intro-to-c$ gcc input_scanf_srt.c
vinicius@laptop:intro-to-c$ ./a.out
Insert a string
Test 123
you entered: Test
```

input_scanf_srt.c

```c
#include <stdio.h>

int main() {
  char my_var[10]; // variable to store the input

  printf("Insert a string\n");
  scanf("%s", my_var); // scanf reading string

  printf("you entered: %s\n", my_var);
}
```

Input using scanf

- Integer example
- Scanf
  - Accept input from
    - String with format
  - Address of variable
  - Arrays variable name is a pointer

```
vinicius@laptop:intro-to-c$ gcc input_scanf_int.c
vinicius@laptop:intro-to-c$ ./a.out
Insert an integer
12323
you entered: 12323
```

input_scanf_int.c

```c
#include <stdio.h>

int main() {
   int my_int; // variable to store the input

   printf("Insert an integer\n");
   // scanf need the variable address "&"
   scanf("%d", &my_int); // scanf reading integer

   printf("you entered: %d\n", my_int);
}
```

Input using formatted strings

- Scanf
  - Formatted input
  - Accepts address of variable (&)
  - Arrays are always pointers

```
vinicius@laptop:intro-to-c$ gcc input_scanf_frm.c
vinicius@laptop:intro-to-c$ ./a.out
Insert two numbers a,b
123,457
you entered: 123 and 457
```

input_scanf_frm.c

```c
#include <stdio.h>

int main() {
   int a; // variable to store the input
   int b; // variable to store the input

   printf("Insert a string\n");
   scanf("%d,%d", &a, &b); // scanf reading two values

   printf("you entered: %d and %d\n", a, b);
}
```

Arrays tests

- Arrays declaration
- Strings == Chars arrays

```
vinicius@laptop:intro-to-c$ gcc array_tests.c
vinicius@laptop:intro-to-c$ ./a.out
chr: A  int: 65
chr: B  int: 66
chr: a  int: 97
chr: b  int: 98
chr:    int: 2
chr: n  int: 110
chr: t  int: 116
chr: *  int: 42
chr:    int: 789
chr:    int: 2
chr:
        int: 10
chr:
        int: 11
```

array_tests.c

```c
#include <stdio.h>

int main(){
   int arr[10] = {0,1,2,110,116,42,789,2,10,11};
   // char arr[10] = "testtestte";

   for (int i=0; i < 10; i++) {
       printf("chr: %c\n", arr[i]);
       printf("int: %d\n", arr[i]);
   }
}
```

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

1.6 Arrays

- Arrays declaration
  - int ndigit[10]
- Array indexing starts with zero
- Counts
  - each digit
  - white spaces
  - other characters
- Uses the representation of numbers to index the array
  - chars = 1 byte integers

Command line ctrl+D sends EOF

```c
#include <stdio.h>

int main() {
    int c, i, nwhite, nother;
    int ndigit[10];

    nwhite = nother = 0;
    for (i = 0; i<10; ++i)
        ndigit[i] = 0;

    while ((c = getchar()) != EOF)
        if (c >= '0' && c <= '9')
            ++ndigit[c-'0'];
        else if (c == ' ' || c == '\n' || c == '\t')
            ++nwhite;
        else
            ++nother;

    printf("digits =");
    for (i = 0; i < 10; ++i)
        printf(" %d", ndigit[i]);
    printf(", white space = %d, other = %d\n", nwhite, nother);
}
```

1.6 Arrays

- Arrays declaration
  - int ndigit[10]
- Array indexing starts with zero
- Counts
  - each digit
  - white spaces
  - other characters
- Uses the representation of numbers to index the array
  - chars = 1 byte integers

Command line ctrl+D sends EOF

```
vinicius@laptop:book-examples$ gcc 1.6arrays.c
vinicius@laptop:book-examples$ ./a.out
teste 123 comp 2003


test
digits = 2 1 2 2 0 0 0 0 0 0, white space = 7, other = 13
```

1.7 Functions

- Subroutine, procedure
- Encapsulate some computation
  - Ignore how is done
  - know what is done
- printf, getchar
- C has no power operator
  - Function power()
  - print 2^i and $(-3)^i$
  - Not practical
  - Only positive powers, small integers
- power(x, y) -> x^y
- Arguments, variable copy, local

1.7functions.c

```c
#include <stdio.h>

int power(int base, int n);

int main() {
    int i;

    for (i=0; i<=10; i++)
        printf("2^%d = %d\n", i, power(2,i));
        // printf("%d %d %d\n", i, power(2,i), power(-3, i));
    return 0;
}

int power(int base, int n) {
    int i, pow;

    pow = 1;
    for (i = 1; i<= n; ++i) {
        pow = pow * base;
    }

    return pow;
}
```

1.7 Functions

- Subroutine, procedure
- Encapsulate some computation
  - Ignore how is done
  - know what is done
- printf, getchar
- C has no power operator
  - Function power()
  - print 2^i and $(-3)^i$
  - Not practical
  - Only positive powers, small integers
- power(x, y) -> x^y
- Arguments, variable copy, local

```
vinicius@laptop:book-examples$ gcc 1.7functions.c
vinicius@laptop:book-examples$ ./a.out
2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
2^4 = 16
2^5 = 32
2^6 = 64
2^7 = 128
2^8 = 256
2^9 = 512
2^10 = 1024
```

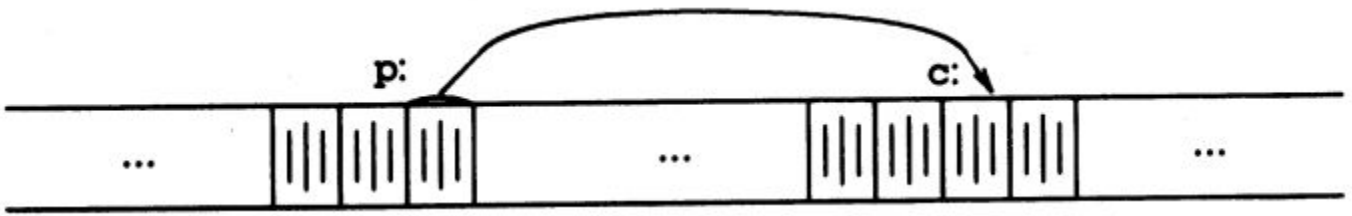
fileinput.c

- 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
```

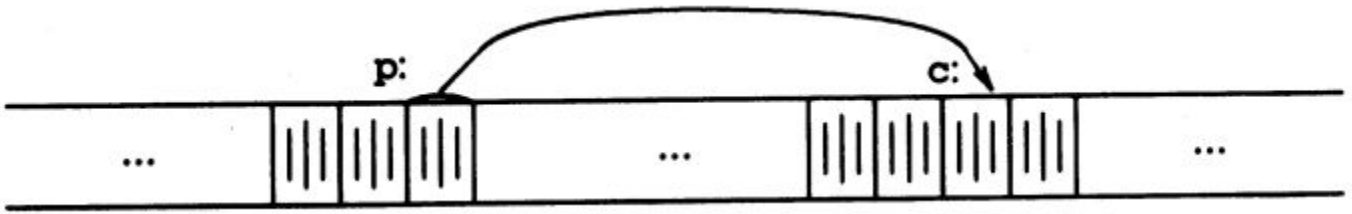
5.1 Pointers

- Memory is an consecutive numbered memory cells
- Pointer is a group of those cells that holds an address
- Operator & gives the address of something p = &c;
- Operator * access the object that pointer is pointing to



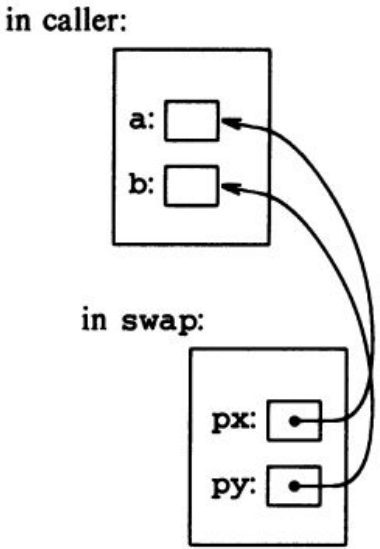
5.1 Pointers

```
int x = 1, y = 2;
int *ip;          // It reads "pointer to an integer"
                  // The expression "*ip" is an int
ip = &x;          // ip receives the address of x
y = *ip;          // copy the content of whatever ip points to y
*ip = *ip + 10;   // *ip affects x
```



5.2 Pointers and function arguments

- C passes arguments to functions by value
- No direct way to alter the variable in the calling function (scope)

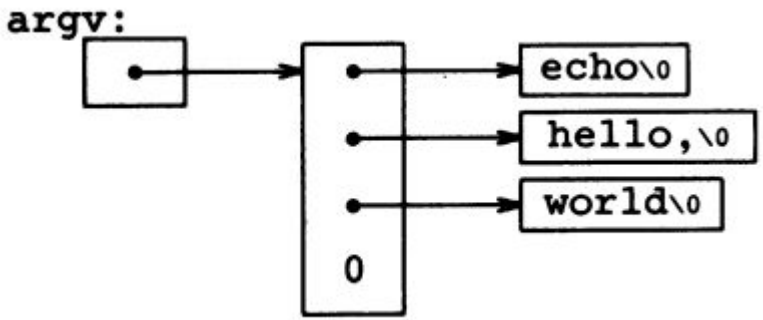



```
int swap(int x, int y) {
    int temp;

    temp = x;
    x = y;
    y = temp;
}

int pswap(int *px, int *py) {
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

Command-line Arguments

commandline_params.c
./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 coord is a tag, we can use that later
    - Has to use struct <tag> <var>; everywhere
  - Typedef Coord is a type (e.g. int)
    - Can be used Coord <var>;

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

struct coord my_coord;
```

```
typedef struct coord {
        int x;
        int y;
} Coord;

Coord my_coord;
```

6.2 Structures and Functions

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

```
struct coord init_coord(int x, int y) {
    struct coord temp_coord;
    temp_coord.x = x;
    temp_coord.y = y;

    return temp_coord;
}

struct coord my_coord = init_coord(2, 3);
```

point-list.c

- Structure with a pointer to the next node

  Coord.next = previous

- typedef
- coordnode is a tag
  - make the struct not unknown, reference after
- typedef define **`CoordNode`** as a type, like int
  - More common

```
typedef struct coordnode {
        struct coordnode *nextcoord;
        int x;
        int y;
} CoordNode;
```

point-list.c

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

```
CoordNode *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->nextcoord;
        if (iter == NULL) {
            printf("\n");
            break;
        }
        printf(" -> ");
    }
```

Questions?

Try examples and code yourself
Really type the examples
Do not copy and paste
https://github.com/vncprado/intro-to-c

Extra material in the next slides

To be continued next class

- Install the environment
  - Test with hello.c
- Try the examples yourself

4.1 Grep

- Split the problem in small parts
- Print the lines that contains a “pattern”
- getline (chapter 1); printf()
- strindex(s, t)
  - Return the index in the string s where t begins
  - -1 if s does not contain t

```
while (there's another line)
    if (the line contains the pattern)
        print  it
```

4.1 Grep

- mygetline
  - Change var s[] to the line read
    - ‘\0’ == End of string
  - While
    - Not limit
    - Not EOF
    - Not “\n”
    - Read next char
  - Return the size of my string
    - Useful later

```
int mygetline(char s[], int lim) {
   int c, i;

   i = 0;
   while (--lim > 0 && (c=getchar()) != EOF && c!= '\n')
       s[i++] = c;
   if (c == '\n')
       s[i++] = c;
   s[i] = '\0'; // End of string
   return i;
}
```

4.1 Grep

- Print the lines that contains a “pattern”
- Split the problem in small parts
- getline (chapter 1); printf()
- strindex(s, t)
  - Return the index in the string s where t begins
  - -1 if s does not contain t

```
int strindex(char s [], char t []) {
    int i, j, k;

    for (i=0; s[i] != '\0'; i++) {
        for (j=i, k=0; t[k] != '\0' && s[j] == t[k]; j++, k++)
            ; // do nothing
        if (k > 0 && t[k] == '\0')
            return i;
    }
    return -1;
}
```

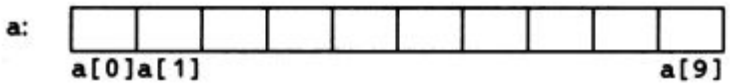
4.1 Grep

- Print the lines that contains a “substring”
- Split the problem in small parts
- getline (chapter 1); printf()
- strindex(s, t)
  - Return the index in the string s where t begins
  - -1 if s does not contain t
- cat loren.txt
- ./a.out < loren.txt

```
int main() {
    char line[MAXLINE];
    int found = 0;

    while (mygetline(line, MAXLINE) > 0)
        if (strindex(line, substring) >= 0) {
            printf("%s", line);
            found++;
        }
    return found;
}
```

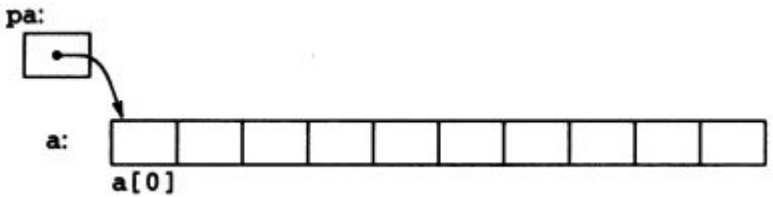
5.3 Pointers and arrays

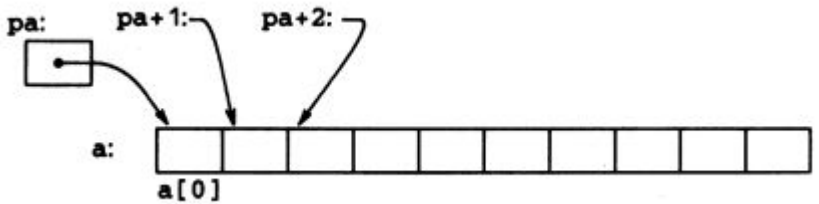
- Strong relationship between pointers and arrays
- Any operation that can be achieved indexing (arr[i]) can also be done with pointers;
- In general it will be faster with pointers



5.3 Pointers and arrays

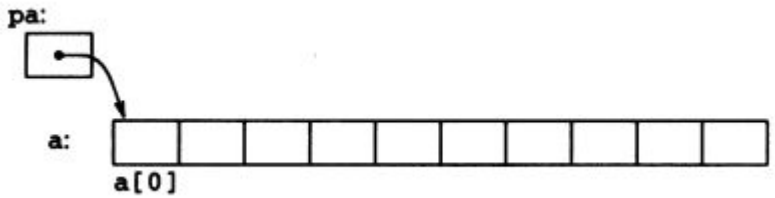
- Strong relationship between pointers and arrays
- Any operation that can be achieved indexing (arr[i]) can also be done with pointers;
- In general it will be faster with pointers

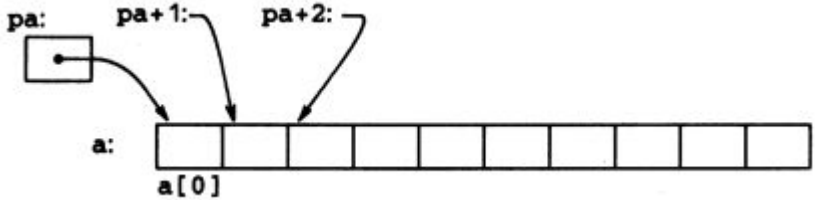




5.3 Pointers and arrays

- 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

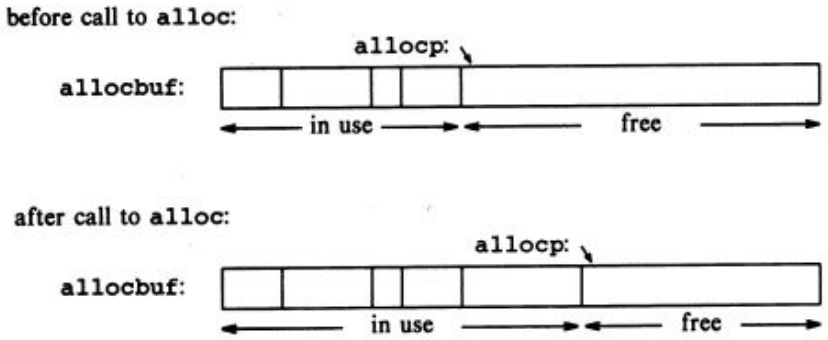




5.4 Address Arithmetic

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

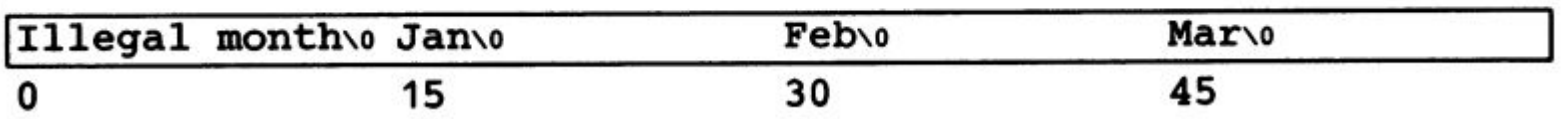
- Pointers are variables themselves
  - Can be stored in arrays, like other variables
- Points to the first letter of the text
- Sorting is faster, just swap pointers
  - 5.6pointerarrays_swap.c
- Complete sort example in the book

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:



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" };
```