

C Survival Guide

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

1

Instant C in 3 slides: Pointers

- Reference operator: `&`
 - address-of
- Dereference operator: `*`
 - contents-of
- Automatic variables
 - Temporary and stored in the stack
- Character pointers: `char* p;`
 - `*p = 0;`
 - contents-of `p` set to 0. (Kaboom!)
- Initialization
 - Initialize a pointer to something before using it. (Doh!)

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

2

Instant C #2: Strings

- Structure
 - C strings are terminated with a null byte
- Functions
 - `strcpy("hello", "world")` will crash
 - `strcmp(s1, s2)` returns 0 if same
- Arguments
 - `argv[0]` is the program name
 - `argv[argc]` is a null pointer

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

3

Instant C #3: Heap Allocation

- Allocation
 - `malloc(bytes)` to reserve heap memory
- Clean up
 - `free(ptr)`
- Static variables
 - `static char* ptr;`
 - Not stored in stack

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

4

Common Causes of 'Death'

1. Uninitialized pointers:
`strcpy(dest, "hello");`
2. C Strings need a null byte at the end
3. Buffer overflow
4. Un-initialized memory
5. Too confident: not checking return values
6. Miss-use of static vs. stack variables.

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

5



Good news: Writing C code is easy!

```
void* myfunction() {
    char *p;
    *p = 0;
    return (void*) &p;
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

6



Bad news: Writing BAD C code is easy!

```
void* myfunction() {
    char *p;
    *p = 0;
    return (void*) &p;
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

7



How do I write good C programs?

- Fluency in C syntax
- Stack vs. Heap
- Key skill: read code for bugs
 - Do not rely solely on compiler warnings, if any, and testing
- C is powerful - it's the System Programmer's choice language

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

8



[The C Language Spirit]

- Made by professional programmers for professional programmers
- Very flexible, very efficient and portable
 - Does not protect the programmers from themselves.
 - Rationale: programmers know what they are doing.
- UNIX and most "serious" system software (servers, compilers, etc) are written in C.
- Can do everything Java and C++ can. It'll just look uglier in C

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

9



[Compiler]

- gcc
 - Preprocessor
 - Compiler
 - Linker
 - See manual "man" for options: man gcc
- "Ansi-C" standards C89 versus C99
 - C99: Mix variable declarations and code (for int i=...)
 - C++ inline comments //a comment
- make – a compilation utility
 - Google 'makefile'

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

10



[Programming in C]

- C = Variables + Instructions

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

11



[Programming in C]

- C = Variables + Instructions

```

| char
| int
| float
| pointer
| array
| string
| ...

```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

12



Programming in C

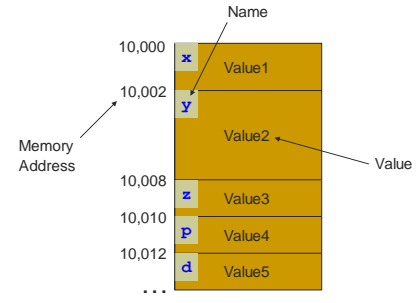
C = Variables + Instructions

char	assignment
int	printf/scanf
float	if
pointer	for
array	while
string	switch

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



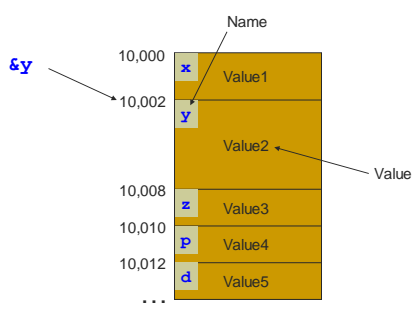
Variables



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



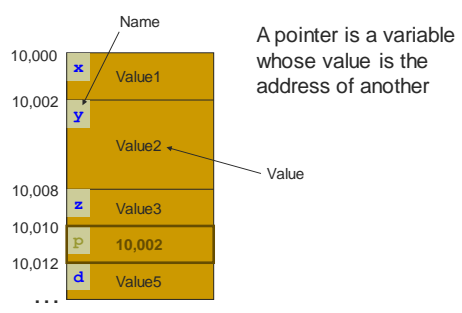
The "&" Operator: Reads "Address of"



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



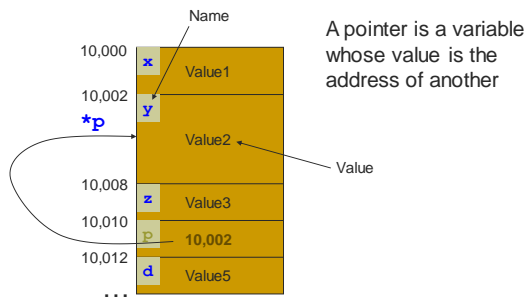
Pointers



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta



The "*" Operator Reads "Variable pointed to by"



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

17



What is the Output?

```
main() {
    int *p, q, x;
    x=10;
    p=&x;
    *p=x+1;
    q=x;
    printf ("Q = %d\n", q);
}
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

18



What is the Output?

```
main() {
    int *p, q, x;
    x=10;
    p=&x;
    *p=x+1;
    q=x;
    printf ("Q = %d\n", q);
}
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

19



What is the Output?

```
main() {
    int *p, q, x;
    x=10;
    p=&x;
    *p=x+1;
    q=x;
    printf ("Q = %d\n", q);
}
```

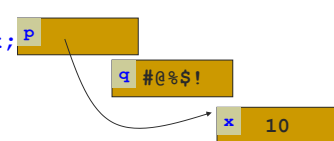
Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

20



[What is the Output?]

```
main() {
  int *p, q, x;
  x=10;
  p=&x;
  *p=x+1;
  q=x;
  printf ("Q = %d\n", q);
}
```



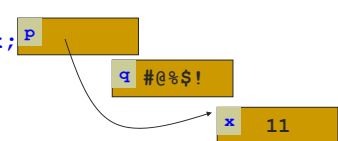
Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

21



[What is the Output?]

```
main() {
  int *p, q, x;
  x=10;
  p=&x;
  *p=x+1;
  q=x;
  printf ("Q = %d\n", q);
}
```



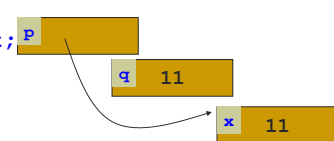
Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

22



[What is the Output?]

```
main() {
  int *p, q, x;
  x=10;
  p=&x;
  *p=x+1;
  q=x;
  printf ("Q = %d\n", q);
}
```



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

23



[Cardinal Rule: Must Initialize Pointers before Using them]

```
int *p;
*p = 10; ← BAD
```

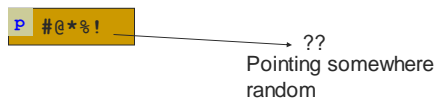
Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

24



Cardinal Rule: Must Initialize Pointers before Using them

```
int *p;
*p = 10;
```



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

25



Cardinal Rule: Must Initialize Pointers before Using them

```
int *p;
*p = 10;
```



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

26



How to Initialize Pointers

- Set pointer equal to location of known variable

```
int *p;
int x;
...
p=&x;
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

27



How to Initialize Pointers

- Use `malloc()`

```
int *p;
...
p=(int*) malloc (sizeof (int));
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

28



How to Initialize Pointers

- Create an Array

```
int p[10];
```

- Same as:

```
int *p;
p=(int*) malloc(10*sizeof(int));
```

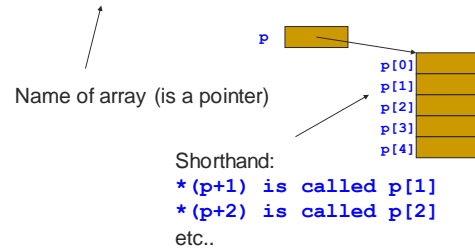
Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

29



Arrays

```
int p[5];
```



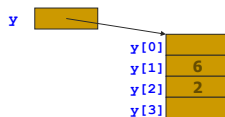
Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

30



Example

```
int y[4];
y[1]=6;
y[2]=2;
```



Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

31



Array Name as Pointer

What's the difference between the examples below

- Example 1:

```
int z[8];
int *q;
q=z;
```

- Example 2:

```
int z[8];
int *q;
q=&z[0];
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

32



Array Name as Pointer

What's the difference between the examples below

- Example 1:
- Example 2:

```
int z[8];
int *q;
q=z;
```

NOTHING!!

```
int z[8];
int *q;
q=&z[0];
```

x (the array name) is a pointer to the beginning of the array, which is &x[0]

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

33

Example

- How much is y at the end:

```
int y, x, *p;
```

```
x = 20;
```

```
*p = 10;
```

```
y = x + *p;
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

34

Example

- How much is y at the end:

```
int y, x, *p;
```

```
x = 20;
```

```
*p = 10;
```

```
y = x + *p;
```

BAD!!
Dereferencing an uninitialized pointer will likely segfault or overwrite something!

Segfault = unauthorized memory access

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

35

Operator	Description	Associativity	
()	Parentheses (function call)	left-to-right	
[]	Brackets (array subscript)		
.	Member selection via object name		
->	Member selection via pointer		
++ --	Postfix increment/decrement	right-to-left	
++ --	Prefix increment/decrement		
+ -	Unary plus/minus		
! ~	Logical negation/bitwise complement		
(type)	Cast (change type)		
*	Dereference		
&	Address		
sizeof	Determine size in bytes		
* / %	Multiplication/division/modulus		left-to-right
+ -	Addition/subtraction		left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right	
< <=	Relational less than/less than or equal to	left-to-right	
> >=	Relational greater than/greater than or equal to		
== !=	Relational is equal to/is not equal to	left-to-right	
&	Bitwise AND	left-to-right	
^	Bitwise exclusive OR	left-to-right	
	Bitwise inclusive OR	left-to-right	
&&	Logical AND	left-to-right	
	Logical OR	left-to-right	
?:	Ternary conditional	right-to-left	
=	Assignment	right-to-left	
+= -=	Addition/subtraction assignment		
*= /=	Multiplication/division assignment		
%= &=	Modulus/bitwise AND assignment		
^= =	Bitwise exclusive/inclusive OR assignment		
<<= >>=	Bitwise shift left/right assignment		
,	Comma (separate expressions)		left-to-right ⁶

Questions

- What's the difference between

```
int* q;
int q[5];
```

- What's wrong with

```
int ptr[2];
ptr[1] = 1;
ptr[2] = 2;
```

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

37



Questions

- What is the value of `b[2]` at the end?

```
int b[3];
int* q;

b[0]=48; b[1]=113; b[2]=1;
q=b;
*(q+1)=2;
b[2]=*b;
b[2]=b[2]+b[1];
```

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

38

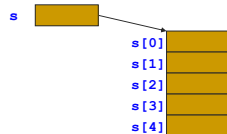


Strings (Null-terminated Arrays of Char)

- Strings are arrays that contain the string characters followed by a "Null" character to indicate end of string.
 - Do not forget to leave room for the null character

- Example

- `char s[5];`



Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

39



Conventions

- Strings
 - "string"
 - "c"
- Character
 - 'c'

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

40



[String Operations]

- `strcpy`
- `strlen`
- `strcat`
- `strcmp`

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

41



[strcpy, strlen]

- `strcpy(ptr1, ptr2);`
 - `ptr1` and `ptr2` are pointers to char
 - `value = strlen(ptr);`
 - `value` is an integer
 - `ptr` is a pointer to char
- ```
int len;
char str[15];
strcpy (str, "Hello, world!");
len = strlen(str);
```

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

42



## [ strcpy, strlen ]

- What's wrong with

```
char str[5];
strcpy (str, "Hello");
```

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

43



## [ strcpy ]

- `strcpy(ptr1, ptr2, num);`
    - `ptr1` and `ptr2` are pointers to char
    - `num` is the number of characters to be copied
- ```
int len;
char str1[15], str2[15];
strcpy (str1, "Hello, world!");
strcpy (str2, str1, 5);
```

Copyright ©: Nahstedt, Angrave, Abdelzاهر, Kravets, Gupta

44



strncpy

- `strncpy(ptr1, ptr2, num);`
 - `ptr1` and `ptr2` are pointers to char
 - `num` is the number of characters to be copied
- ```
int len;
char str1[15],
 str2[15];
strcpy(str1,
 "Hello, world!");
strncpy(str2,
 str1, 5);
```

Caution: `strncpy` blindly copies the characters. It does not voluntarily append the string-terminating null character.

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

45



## strcat

- `strcat(ptr1, ptr2);`
  - `ptr1` and `ptr2` are pointers to char
- Concatenates the two null terminated strings yielding one string (pointed to by `ptr1`).

```
char S[25] = "world!";
char D[25] = "Hello, ";
strcat(D, S);
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

46



## strcat Example

- What's wrong with

```
char S[25] = "world!";
strcat("Hello, ", S);
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

47



## strcmp

- `diff = strcmp(ptr1, ptr2);`
  - `diff` is an integer
  - `ptr1` and `ptr2` are pointers to char
- Returns
  - < 0 if `ptr1` is less than `ptr2` (earlier in a dictionary)
  - 0 if `ptr1` is the same as `ptr2`
  - > 0 if `ptr1` is greater than `ptr2` (later in a dictionary)

```
int diff;
char s1[25] = "pat";
char s2[25] = "pet";
diff = strcmp(s1, s2);
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

48



## Math: Increment and Decrement Operators

- Example 1:

```
int x, y, z, w;
y=10; w=2;
x=++y;
z=--w;
```

- Example 2:

```
int x, y;
y=10; w=2;
x=y++;
z=w--;
```

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

49



## Math: Increment and Decrement Operators

- Example 1:

```
int x, y, z, w;
y=10; w=2;
x=++y;
z=--w;
```

- Example 2:

```
int x, y;
y=10; w=2;
x=y++;
z=w--;
```

- First increment/decrement, then assign result
- x** is 11, **z** is 1

- First assign result, then increment/decrement
- x** is 10, **z** is 2

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

50



## Math: Increment and Decrement Operators on Pointers

- Example 1:

```
int a[2];
int number1, number2, *p;
a[0]=1; a[1]=10; a[2]=100;
p=a;
number1 = *p++;
number2 = *p;
```

- What will **number1** and **number2** be at the end?

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

51



## Math: Increment and Decrement Operators on Pointers

- Example 1:

```
int a[2];
int number1, number2, *p;
a[0]=1; a[1]=10; a[2]=100;
p=a;
number1 = *p++;
number2 = *p;
```

Hint: ++ increments pointer **p** not variable **\*p**

- What will **number1** and **number2** be at the end?

Copyright ©: Nahstedt, Angrave, Abdelzaker, Kravets, Gupta

52



## Logic: Relational (Condition) Operators

|    |                          |
|----|--------------------------|
| == | equal to                 |
| != | not equal to             |
| >  | greater than             |
| <  | less than                |
| >= | greater than or equal to |
| <= | less than or equal to    |

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

53



## Logic Example

```
if (a == b)
 printf ("Equal.");
else
 printf ("Not Equal.");
```

- Question: what will happen if I replaced the above with:

```
if (a = b)
 printf ("Equal.");
else
 printf ("Not Equal.");
```

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

54



## Review

- `int p1;`  
What does `&p1` mean?

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

55



## Review

- `char**argv;`  
What type is `argv`?  
What type is `*argv`?  
What type is `**argv`?

Copyright ©: Nahrstedt, Angrave, Abdelzaher, Kravets, Gupta

56



## [ Review ]

- `main(int argc, char**argv) {`  
what is `*argv`?  
what is `argv[argc]`?

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

57



## [ Review ]

- What are the differences between `x` and `y`?  
`char* f() {`  
    `char *x;`  
    `static char*y;`  
    `return y;`  
}

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

58



## [ Review: Debugging ]

```
if(strcmp("a", "a"))
 printf("same!");
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

59



## [ Review: Debugging ]

```
int i = 4;
int *iptr;
iptr = &i;
*iptr = 5;//now i=5
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

60



## [ Review: Debugging ]

```
char *p;
p=(char*)malloc(99);
strcpy("Hello",p);
printf("%s World",p);
free(p);
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

61



## [ Review: Debugging ]

```
char msg[5];
strcpy (msg,"Hello");
```

Copyright ©: Nahstedt, Angrave, Abdelzaher, Kravets, Gupta

62

