Strings in *C*

Professor Hugh C. Lauer
CS-2303, System Programming Concepts

(Slides include materials from *The C Programming Language*, 2nd edition, by Kernighan and Ritchie, *Absolute C++*, by Walter Savitch, *The C++ Programming Language,* Special Edition, by Bjarne Stroustrup, and from *C: How to Program*, 5th and 6th editions, by Deitel and Deitel)

Reading Assignment

- **Kernighan & Ritchie, Chapter 5**
  - *All the way through!*
- ***Study* §5.5 in detail — pages 104-107**
  - Character pointer and functions
  - You will use these *all the time!*
- ***Study* §5.10 — pages 114-118**
  - Command line arguments
  - You will use these *a lot!*
  - Applicable also to *C++*

(Skip ahead)

Review

- ***Array* – a set of *elements* all of the same type stored contiguously in memory – e.g.,**
  - *int A[25]; // 25 integers*
  - *struct Str B[15]; /* 15 objects of type struct Str */*
  - *double C[]; /* indeterminate # of doubles */*

- ***Pointer* – a variable whose value is the *location* of some other object**
  - *float *p; // pointer to a float*

Review (continued)

int A[25];   // 25 integers

- ***Type* of *A[i]* is *int* (for all *i*).**

- ***Type* of *A* is *int * const***
    - I.e., constant pointer to *int*

Summary

- Arrays and pointers are *closely* related
- Let *int A[25];*
  *int *p; int i, j;*
- Let *p = A;*
- Then *p* points to *A[0]*
  p + i points to *A[i]*
  &A[j] == p+j
  **(p+j)* is the same as *A[j]*

Summary (continued)

- **If *void f(int A[], int arraySize);***
- **Then *f(&B[i], bSize-i)* calls *f* with subarray of *B* as argument**
    - Starting at element *i*, continuing for *bSize-i* elements

Review (concluded)

void f(int A[], int arraySize);

and

void f(int *A, int arraySize);

are identical!

- **Most *C* programmers use pointer notation rather than array notation**
  - In these kinds of situations

Additional Notes

- **A pointer is *not* an integer …**
  - Not necessarily the same size
  - May not be assigned to each other
- **… except for value of zero!**
  - Called *NULL* in *C*; defined in *<stdio.h>*
  - Means "pointer to nowhere"
- ***void ** is a pointer to no type at all**
  - May be assigned to *any* pointer type
  - *Any* pointer type may be assigned to *void **

Questions?

Characters in C

C99 & C++ introduce a new data type called *wchar* for international text

- ***char*** **is a one-byte data type capable of holding a character (mostly printable)**
  - Treated as an arithmetic integer type
  - (Usually) *unsigned*
- **May be used in arithmetic expressions**
  - Add, subtract, multiply, divide, etc.
- **Character constants**
  - *'a', 'b', 'c', …'z', '0', '1', … '9', '+', '-', '=', '!', '~'*, etc, *'\n', '\t', '\0'*, etc.
  - *A-Z, a-z, 0-9* are *in order*, so that arithmetic can be done

Strings in C

- ***Definition:–*** **A *string* is a character array ending in the *null character '\0'* — i.e.,**
  - *char s[256];*
  - *char t[] = "This is an initialized string!";*
  - *char *u = "This is another string!";*
- ***String constants*** **are in double quotes *"like this"***
  - May contain any characters
  - Including *\"* and *\'* — see p. 38, 193 of K&R
- **String constants may not span lines in code**
  - However, they may be concatenated — e.g.,
  - *"Hello, " "World!\n"* is the same as *"Hello, World!\n"*

Strings in C (continued)

- **Let**
  - *char *u = "This is another string!";*
- **Then**
  - *u[0] == 'T'*
    u[1] == 'h'
    u[2] == 'i'
    …
    u[20] == ‘n'
    u[21] == 'g'
    u[22] == '!'
    u[23] == '\0'

Support for Strings in C

- **Most string manipulation is done through functions in <*string.h*>**
- **String functions *depend* upon final *'\0'***
    - So you don't have to count the characters!
- **Examples:–**
    - *int strlen(char *s)* – returns length of string
        - Excluding final *'\0'*
    - *char* strcpy(char *s, char *ct)* – Copies string *ct* to string *s*, return *s*
        - *s* must be big enough to hold contents of *ct*
        - *ct* may be smaller than *s*

Additional String Functions

- ***int strcmp(char *s, char *t)***
  - lexically compares *s* and *t*, returns *<0* if *s < t*, *>0* if *s > t*, and zero if *s* and *t* are identical
  - K&R p. 106 for exact details expressed in code
- ***char* strcat(char *s, char *ct)***
  - Concatenates string *ct* to onto end of string *s*, returns *s*
  - *s* must be big enough to hold contents of both strings!
- **Other string functions**
  - *strchr(), strrchr(), strspn(), strcspn() strpbrk(), strstr(), strtok(),* …
- **See K&R**
  - pp. 105-106 for various implementations
  - §B.3 for complete list and specifications

Character functions in C

- See *<ctype.h>*

- These return either *0* (i.e., *false*) or *1* (i.e., *true*)

int isdigit(int c)	*int isalpha(int c)*
int isalnum(int c)	*int isxdigit(int c)*
int islower(int c)	*int isupper(int c)*
int isspace(int c)	*int iscntrl(int c)*
int ispunct(int c)	*int isprint(int c)*
int isgraph(int c)	

- These change case (if appropriate) and return characters

int toupper(int c)	*int tolower(int c)*

String Conversion Functions in C

- See *<stdlib.h>*
  - K&R p. 251-252

double atof(const char *s)

int atoi(const char *s)

long atol(const char *s)

double strtod(const char *s, char **endp)

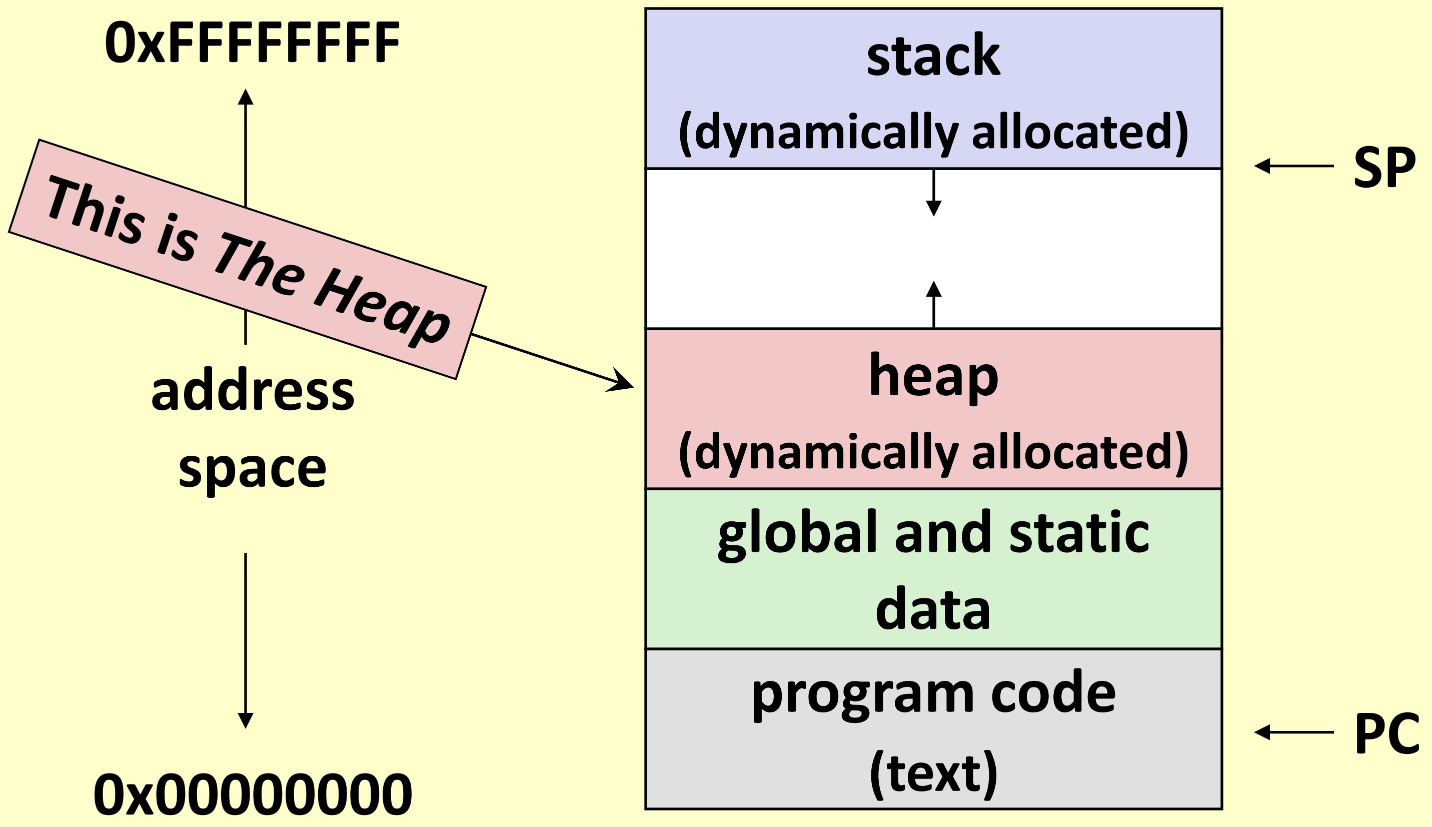
long strtol(const char *s, char **endp, int base)

unsigned long strtoul(const char *s, char **endp, int base)

Dilemma

- **Question:–**
  - If strings are arrays of characters, …
  - and if arrays cannot be returned from functions, …
  - how can we manipulate variable length strings and pass them around our programs?
- **Answer:–**
  - Use storage allocated in *The Heap!*

Dynamic Data Allocation



Typical Usage with Strings

```
char *getTextFromSomewhere(…);

int main(){
    char *txt;
    …;
    txt = getTextFromSomewhere(…);
    …;
    printf("The text returned is %s.", txt);
    free(txt);
}
```

getTextFromSomewhere() cannot return pointer to automatic string variable within its own scope!

Why not?

Typical Usage (continued)

```
char *getTextFromSomewhere (…)
    {
    char *t;
    ...
    t = malloc(stringLength);
    ...    /* fill in string t*/
    return t;
}
```

```
int main(){
    char * txt;
    …;
    txt = getTextFromSomewhere
    (…);
    …;
    printf("The text returned is
    %s.", txt);
    free(txt);
```

Don't forget to free() the returned string!

Questions?

String Manipulation in C

- **Almost all *C* programs that manipulate text do so with *malloc*'ed and *free*'d memory**
- **No limit on size of string in *C***
- ***You* need to be aware of sizes of character arrays!**
- ***You* need to remember to free storage when it is no longer needed**
    - *Before* forgetting pointer to that storage!

Input-Output Functions

- ***printf(const char *format, ...)***
  - Format string may contain *%s* – inserts a string argument (i.e., *char **) up to trailing '*\0*'
- ***scanf(const char *format, ...)***
  - Format string may contain *%s* – scans a string into argument (i.e., *char **) up to next “white space”
  - Adds '*\0*'
- **Related functions**
  - *fprintf(), fscanf()* – to/from a file
  - *sprintf(), sscanf()* – to/from a string

Hazard with scanf()

```
char word[20];
…;
scanf("%s", word);
```

- *scanf* **will continue to scan characters from input until a** ***space*****,** ***tab*****,** ***new-line*****, or** ***EOF*** **is detected**
  - An *unbounded* amount of input
  - May overflow allocated character array
  - Probable corruption of data!
  - *scanf* adds trailing *'\0'*
- **Solution:–**

```
scanf("%19s", word);
```

Questions?

Command Line Arguments

- **See §5.10**
- **By convention, *main* takes two arguments:-**

  *int main(int argc, char *argv[]);*

  - *argc* is number of arguments
  - *argv[0]* is string name of program itself
  - *argv[i]* is argument *i* in string form
    - i.e., *i < argc*
  - *argv[argc]* contains a null pointer!
- **Sometimes you will see (the equivalent)**

  *int main(int argc, char **argv);*

An array of pointers to *char* (i.e., strings)

Example — PA #2

- Instead of *prom* ser for game parameters, simply take them from command line

```
 % life 100 50 1000
```

would play the Gam ife on a grid of 100 × 50 squares for 1000 generatio

argv[0] – name of program

argv[1] – first program argument

Example — PA #2 (continued)

```
int main(int argc, char *argv[])){
    int xSize, ySize, gens, j;
    char grid[][];
    if (argc <= 1) {
        printf("Input parameters:-
    ");
        scanf("%d%d%d", &xSize,
    &ySize, &gens);
    } else {
        xSize = atoi(argv[1]);
        ySize = atoi(argv[2]);
        gens = atoi(argv[3]);
    }
```

```
    grid = calloc(xSize, sizeof (char *);
    for (j = 0; j < xSize; j++)
        grid[j] = calloc(ySize,
                sizeof (char));

    /* rest of program using grid[i][j]
    */

    for (j = 0; j < xSize; j++)
        free(grid[j];
    free(grid);
    return 0;
}   //  main(argc, argv)
```

Example — PA #2 (continued)

```
int main(int argc, char *argv[])){
    int xSize, ySize, gens, j;
    char grid[][];
    if (argc <= 1) {
        printf("Input parameters:-
");
        scanf("%d%d%d", &xSize,
&ySize, &gens);
    } else {
        xSize = atoi(argv[1]);
        ySize = atoi(argv[2]);
        gens = atoi(argv[3]);
    }
```

```
grid = calloc(xSize, sizeof (char *);
for (j = 0; j < xSize; j++)
    grid[j] = calloc(ySize,
                sizeof (char));

/* rest of program using grid[i][j]
*/

for (j = 0; j < xSize; j++)
    free(grid[j];
}
```

Convert argument #1 to
int for *xSize*

int for *ySize*

int for *gens*

Questions?