CS 420 Advanced Programming Languages
Fall Semester, 2022
Doc 25 C 2
Dec 1 2022 (updated)

Arrays

```
#include <stdio.h>
int main (void){
    int a[5]  = {10, 20, 30, 40};
    int b[5];
    printf("%i\n", a[0]);              10
    printf("%i\n", *a);                10
    printf("%i\n", a[2]);              30
    printf("%i\n", *(a + 2));          30
    printf("%i\n", a[4]);              0
    printf("%i\n", *(a + 4));          0
    printf("%i\n", a[40]);             -1794897457
    printf("%i\n", *(a + 40));         -1794897457
    printf("%i\n", a[-2]);             83525728
    printf("%i\n", *(a - 2));          83525728
    printf("%i\n", b[0]);              1
    printf("%i\n", b[2]);              1
    printf("%i\n", b[20]);             0
}
```

Overriding other Variables

```
int main (void)
{
    int c = 42;
    int a[5];
    int b = 42;
    int d = 42;
    for (int i = -10; i < 20; i++)
        a[i] = i;
    printf("%i\n",b);          -3
    printf("%i\n",c);          -2
    printf("%i\n",d);          -4
}
```

3

Arrays are not Values

int[5] just declares a pointer int*

```
int* foo() {
    int bar[5] = {1,2,3,4};
    return bar;
}
int main (int argc, char *argv[]) {
    int test[5];
    test = foo();

    return(0);
}
```

warning: address of stack memory associated with local variable 'bar' returned

error: array type 'int[5]' is not assignable

Multi-Dimensional Arrays

```
int  M[4][5];              4 rows and 5 columns
```

```
int  M[4][5] = {
                  { 10,  5, -3, 17, 82 },
                  {  9,  0,  0,  8, -7 },
                  { 32, 20,  1,  0, 14 },
                  {  0,  0,  8,  7,  6 }
                };
```

```
int  M[4][5] = { 10, 5, -3, 17, 82, 9, 0, 0, 8, -7, 32,
                 20, 1, 0, 14, 0, 0, 8, 7, 6 };
```

Dynamic Arrays C11

```
#include <stdio.h>

int main (void)
{
    int size;
    printf("Enter the size");
    scanf("%i",&size);

    int example[size];
    int b = -12;
    for (int i=0; i < size; i++)
            example[i] = i*10;
    printf("%i\n", example[2]);
    printf("%i\n", b);
}
```

Strings are null-terminated Arrays

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

int main (void)
{
    char a[ ] = { "Hello"};
    char b[ ] = "Hello";
    char c[ ] = {'H','e','l','l','o','\0'};
    printf("%s\n", b);
    assert( a[0] == 'H');
    assert( *(a + 4) == 'o');
    assert( a[5] == '\0');
    int length = strlen(a);  // counts character up to \0
    assert(length == 5);
    assert(strcmp(a, b) == 1);
}
```

a, b, c are different ways to define the same thing

Scanf buffer overflow

```
#include <stdio.h>
#include <assert.h>

int main (void)
{
    char string[5];
    printf("Enter a string");
    scanf("%s",string);
    printf("\n string is %s",string );
}
```

Input

```
Enter a string
abcdefghijk

 string is abcdefghijk
```

Preventing the Overflow

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

int main (void)
{
    char string[5];
    printf("Enter a string");
    scanf("%5s",string);
    printf("\n string is %s",string );
}
```

```
Enter a string
abcdefghijk

 string is abcde
```

stdio.h

clearerr()	clrmemf()	fclose()	fdelrec()	feof()
ferror()	fflush()	fgetc()	fgetpos()	fgets()
fldata()	flocate()	fopen()	fprintf()	fputc()
fputs()	fread()	freopen()	fscanf()	fseek()
fseeko()	fsetpos()	ftell()	ftello()	fupdate()
fwrite()	getc()	getchar()	gets()	perror()
printf()	putc()	putchar()	puts()	remove()
rename()	rewind()	scanf()	setbuf()	setvbuf()
sprintf()	sscanf()	svc99()	tmpfile()	tmpnam()
ungetc()	vfprintf()	vprintf()	vsprintf()	

fgets - Read Line of text from a file

```
#include <stdio.h>

int main () {
   FILE *fp;
   char str[60];

   fp = fopen("file.txt" , "r");
   if(fp == NULL) {
      perror("Error opening file");
      return(-1);
   }
   if( fgets (str, 60, fp)!=NULL ) {
      puts(str);
   }
   fclose(fp);

   return(0);
}
```

argc, argv

```
#include <stdio.h>
#include <assert.h>
#include <string.h>

int main (int argc, char *argv[])
{
    printf("Number of arguments = %i\n", argc);
    printf("Name of the function = %s\n", argv[0]);
    for (int k = 1; k < argc; k++)
        printf("Argument %i is %s\n", k , argv[k]);
}
```

./a.out cat rat mat

Number of arguments = 4
Name of the function = ./a.out
Argument 1 is cat
Argument 2 is rat
Argument 3 is mat

./a.out -f sample.txt

Number of arguments = 3
Name of the function = ./a.out
Argument 1 is -f
Argument 2 is sample.txt

```c
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[]) {
    if (argc < 2) {
        printf("Missing file name");
        return 1;
    }
    char fileName[10];
    strncpy(fileName,argv[1], 10);
    FILE *fp;
    char str[60];

    fp = fopen(fileName , "r");
    if(fp == NULL) {
        perror("Error opening file");
        return(-1);
    }
    if( fgets (str, 60, fp)!=NULL ) { puts(str); }
    fclose(fp);

    return(0);
}
```

strncpy verses strcpy

strncpy(dest, src, size)

- Requires max number of characters to copy
- Eliminates buffer overflow

strcpy(dest, src)

- Will copy all the characters from src to dest
- Can overflow dest

There are a number of such function pairs in C

Strings are null terminated

```
int  stringLength (const char  string[])
{
    int  count = 0;

    while ( string[count] != '\0' )
        ++count;

    return count;
}
```

Buffers

```
void  readLine (char  buffer[]){
    char  character;
    int   i = 0;

    do {
        character = getchar ();
        buffer[i] = character;
        ++i;
    }
    while ( character != '\n' );

    buffer[i - 1] = '\0';
}
```

```
int main (void) {
    int   i;
    char  line[81];
    void  readLine (char  buffer[]);

    for ( i = 0; i < 3; ++i ) {
        readLine (line);
        printf ("%s\n\n", line);
    }

    return 0;
}
```

gets() reads a line

readLine

- shows how to implement gets()
- A common operation in C - buffer

Structures

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

```
struct date  today;
today.month = 9;
today.day = 25;
today.year = 2015;

struct date  purchaseDate = {11,29,2022};
struct date  registrationDate = {.month = 11, .year = 2022, .day = 30};
```

Struct & Functions

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

struct date  dateUpdate (struct date  quiz) {
    quiz.day = 35;
    return quiz;
}
```

structs are copied when passed as argument

```
int main (int argc, char *argv[]) {
    struct date today = {11,30,2022};
    struct date modifiedDate = dateUpdate(today);
    printf("today day %i\n", today.day);
    printf("modifiedDate day %i\n", modifiedDate.day);
    printf("today %i\n", today);
    return(0);
}
```

today day 30
modifiedDate day 35
today 11

Pointers * &

```
int main (int argc, char *argv[]) {
    int   count = 10, x;
    int   *countPointer;

    countPointer = &count;
    x = *countPointer;
    assert(x == 10);
    *countPointer = 20;
    assert(count == 20);
    return 0;
}
```

*

dereference

&

Address of

Pointer Pointer Pointer

```
int main (int argc, char *argv[]) {
    int   count = 10, x;
    int   *countPointer;
    int   **countPointerPointer;
    int   ***countPointerPointerPointer;

    countPointer = &count;
    countPointerPointer = &countPointer;
    countPointerPointerPointer = &countPointerPointer;

    x = ***countPointerPointerPointer;
    assert(x == 10);

    ***countPointerPointerPointer = 20;
    assert(x == 10);
    assert(count == 20);
    return 0;
}
```

How You know You Really Messed Up

Process finished with exit code 139 (interrupted by signal 11: SIGSEGV)

Dynamic (Heap) Memory

malloc

Allocates a block of memory from Heap

a = malloc(10 * sizeof(int))

calloc

Allocates a block of memory from Heap

Zeros out the memory

a = calloc(10, sizeof(int))

free

releases memory back to heap

free(a)

Pointers, Arrays, Functions

```
int* foo() {
    int bar[5] = {1,2,3,4};
    return bar;
}
int main (int argc, char *argv[]) {
    int test[5];
    test = foo();

    return(0);
}
```

```
int* foo() {
    int bar[5] = {1,2,3,4};
    return bar;
}
int main (int argc, char *argv[]) {
    int *test;
    test = foo();
    printf("%i",test[2]);
    return(0);
}
```

Pointers, Arrays, Functions

```
int* foo() {
    int *bar = calloc(5, sizeof(int));
    for (int k = 0; k < 5; k++)
        bar[k] = k;
    return bar;
}

int main (int argc, char *argv[]) {
    int *test;
    test = foo();
    printf("%i",test[2]);
    free(test);
    return(0);
}
```

Struct & Dynamic Memory

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

struct  date*  dateUpdate (struct date  *quiz) {
    (*quiz).day = 35;
    return quiz;
}

 int main (int argc, char *argv[]) {
    struct date *today = (struct date*) calloc(1, sizeof(struct date));
    struct date *modifiedDate = dateUpdate(today);
    printf("today day %i\n", today->day);
    printf("modifiedDate day %i\n", modifiedDate->day);
    printf("today %i\n", today);
    return(0);
}
```

```
today day 35
modifiedDate day 35
today 26787888
```

Where does the * go?

struct date *dateUpdate (struct date* quiz) {

struct date * dateUpdate (struct date * quiz) {

struct date* dateUpdate (struct date *quiz) {

struct date* dateUpdate (struct date* quiz) {

struct date *today

struct date * today

struct date*   today

C Fun

```
struct Person {
    int age;
    char name[250];
};

struct Person * getPerson(){
    char name[5] = "John";
    struct Person *p;
    p = (struct Person *) malloc(sizeof(struct Person));
    strcpy(p->name, name);
    p->age = 20;
    return p;
}
```

What if don't want to always allocate fixed space

Trouble

```
struct Person {
    int age;
    char *name;
};

struct Person * getPerson(){
    char name[5] = "John";
    struct Person *p;
    p = (struct Person *) malloc(sizeof(struct Person));
    strcpy(p->name, name);
    p->age = 20;
    return p;
}
```

Process finished with exit code 139 (interrupted by signal 11: SIGSEGV)

p->name does not point anywhere,

Done Correctly

```
struct Person {
    int age;
    char *name;
};

struct Person * getPerson(){
    char name[5] = "John";
    struct Person *p;
    p = (struct Person *) malloc(sizeof(struct Person));
    p->name = malloc(sizeof(name));
    strcpy(p->name, name);
    p->age = 20;
    return p;
}
```