COMP 2355 Introduction to Systems Programming

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

Today

- Pointer Arithmetic

Printing Pointers

```
void * p = malloc(42);
printf("%p", p);
```

Pointer Addition

```
int  * ip = malloc(42);
printf("%p %p %p", ip, ip + 1, ip + 2);
char * cp = malloc(42);
printf("%p %p %p", cp, cp + 1, cp + 2);
long long * lp = malloc(42);
printf("%p %p %p", lp, lp + 1, lp + 2);
```

The Rule

Adding “1” to a pointer increments the address by the size of the base-type.

Pointer Addition

```
char ** cpp = malloc(42);
printf("%p %p %p", cpp, cpp + 1, cpp - 1);
struct stat * sp = malloc(42);
printf("%p %p %p", sp, sp + 1, sp + 2);
void * vp = malloc(42);
printf("%p %p %p", vp, vp + 1, vp + 2);
```

Addresses and Arrays

```
int * ip = malloc(sizeof(int)*42);
printf("%p %p %p %p", ip, &ip[0], &ip[1], &ip[2]);
char* cp = malloc(sizeof(char)*42);
printf("%p %p %p %p", cp, &cp[0], &cp[1], &cp[2]);
long long lp[42];
printf("%p %p %p %p", lp, &lp[0], &lp[1], &lp[2]);
```

Arrays

A variable refering to an array is a pointer to the 0-th element of the array.

Arrays and Pointer Arithmetic

```
int * ip = malloc(sizeof(int)*42);
*ip = 10;
*(ip+1) = 11;
printf("%d %d", ip[0], ip[1]);
```

Arrays and Pointer Arithmetic

```
int  * ip = malloc(sizeof(int)*42);
char * cp = (char*) ip;
printf("%p %p %p %p", &ip[0], &ip[1], &cp[0], &cp[1]);
cp[0] = 1;
cp[1] = 1;
cp[2] = 1;
cp[3] = 1;
printf("%x", (unsigned int) ip[0]);
```

Example: `strlen`

```
size_t strlen(const char * s) {
  const char * p = s;
  while (*p) p++;
  return p - s;
}
```

Example: `strlen`

```
size_t strlen(const char * s) {
  size_t p = 0;
  while (s[p] != '\0') p++;
  return p;
}
```

Exercise

Write your own implementation of `strcpy` (call it `mystrcpy` to avoid naming conflicts with libc). Also implement a `main` method to check that it works.

Exercise

Write your own implementation of `strcat` (call it `mystrcat` to avoid naming conflicts with libc). Also implement a `main` method to check that it works.

More fun... (1/3)

```
struct data {
  int foo;
  float bar;
};
struct data * dp = malloc(sizeof(struct data) + 12);
dp->foo = 42;
dp->bar = 4.2;
strcpy((char*) &dp[1], "Hello World");
```

More fun... (2/3)

```
struct data {
  int foo;
  float bar;
};
struct data * dp = malloc(sizeof(struct data) + 12);
dp->foo = 42;
dp->bar = 4.2;
strcpy((char*) (dp+1), "Hello World");
```

More fun... (3/3)

```
struct data {
  int foo;
  float bar;
};
struct data * dp = malloc(sizeof(struct data) + 12);
dp->foo = 42;
dp->bar = 4.2;
strcpy((char*) dp+1, "Hello World");
```

Exercise

```
static const char * fun(const char * ch) {
  ch += 3;
  return ch;
}
int main(int argc, char ** argv){
  const char * str = "string";
  const char * cptr = fun(str);
  printf("%s\n",cptr);
  return 0;
}
```

Yes, we can!

```
int * ip = malloc(sizeof(int) * 10);
int * mp = &ip[5];
mp[-5] = 42;
printf("%d", ip[0]);
```

Yes, we can!

```
int * ip = malloc(sizeof(int) * 5);
struct {
  float f;  int j;
} * sp = (void*) ip;
memset(ip, 1, sizeof(int) * 5);
sp->j = 42;
sp->f = 4.2;
sp[1].j = 62;
printf("%d %d %d %d %d",
       ip[0], ip[1], ip[2], ip[3], ip[4]);
```

Good night, and good luck

```
int * ip = malloc(sizeof(int) * 10);
ip[-1] = 4;
ip[10] = 4;
```

About NULL

- `NULL` of type `void*` corresponds to "null" in Java
- `malloc(0)` is illegal on some systems, returns `NULL` on some and a valid address in memory that is not equal to any existing object on others. Avoid it!

Exercise

Write a program that gets a string from the user and checks whether or not it is a palindrome. Hint: use `fgets` or `argv` to obtain the input.

Example for a palindrome: "abbcbba"

Exercise

Implement a function that receives a string as input and returns 1 if it's a palindrome, 0 otherwise, ignoring non-letter signs and letter case.

For example, "Madam, I'm Adam" should be declared a palindrome.

Questions

?