CSCI 2132
Software Development

Lecture 23:

Pointers and Arrays (Pointer Arithmetic)

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Review of matrial from Lab 7:
  - Introduction to ‘make’ and Makefile
  - Review of history of Version Control Systems (rcs, cvs, Subversion, git)
  - Introduction to git, github, and GitLab
- Review of pointers
- Finished statistics.c example

Using `const` to Protect Arguments

- Passing pointers as arguments is usually done for function to make change to the caller variables
- Another reason: efficiency
- We may want to prevent accidental change to the arguments
- Example:

```
void f(const int *p) {
  /* The function is not allowed
     to modify *p */
}
```

Pointers and Arrays

- In C, pointers and arrays are closely related
- Array name is a pointer to 0th element of the array
- This is why an array argument is passed as a pointer
- Example:

```
int a[10];
int *p = &a[0];
```

- Equivalent to:

```
int *p = a;
```

Pointer Arithmetic: Pointer + Integer

- We can **add integer to a pointer:**
  - If `p` points to `a[i]`, `p+j` points to `a[i+j]`
- Example:

```
1: int a[10] = {9};
2: int *p = &a[1];
3: (*(p+3))++;
4: printf("%d %d\n", a[1], a[4]);
```

- What is the output of this program?

Pointer Arithmetic: Subtraction

- We can **subtract integer from a pointer:**
  - If `p` points to `a[i]`, then `p-j` points to `a[i-j]`
- We can **subtract pointers:**
  - If `p` points to `a[i]` and `q` points to `a[j]`, then `p-q` is `j-i`
- Example:

```
int a[10];
int *p = &a[0];
int *q = &a[5];
printf("%d\n", p-q);
```

- What is the output?

Pointer Comparison

- If pointers `p` and `q` point to elements of the same array `a[i]` and `a[j]` then
  - if `i < j` $\Rightarrow$ `p < q`
  - if `i == j` $\Rightarrow$ `p == q`
  - if `i > j` $\Rightarrow$ `p > q`
- What if we compare pointers or subtract pointers that do not point to the elements of the same array?

Pointer Comparison

- If pointers `p` and `q` point to elements of the same array `a[i]` and `a[j]` then
  - if `i < j` $\Rightarrow$ `p < q`
  - if `i == j` $\Rightarrow$ `p == q`
  - if `i > j` $\Rightarrow$ `p > q`
- What if we compare pointers or subtract pointers that do not point to the elements of the same array?
- *Undefined behaviour*

More Equivalent Statements

- `a[2] = 4;` and `*(a+2) = 4;`
- `*(p+3) = 5;` and `p[3] = 5;`
- Code:

```
int i;
for (i = 0; i < 10; i++)
    a[i] = 0;
```

- and

```
for (p = a; p < &a[10]; p++)
    *p = 0;
```

. . . continued

- Also equivalent loop:

```
for (p = a; p < a+10; p++)
    *p = 0;
```

- A difference between array name and pointer: cannot change array name value (i.e., array location)
- Note: `++` and `--` have higher precedence than `*`
- I.e., `*p++` means `*(p++)` rather than `(*p)++`
- Array parameters can be expressed as pointers, e.g.:

```
int max_array(int *a, int len);
```

Efficiency of Pointers vs. Arrays

- Pointer arithmetic has been generally more efficient
- However, modern compilers optimize subscripts to be as efficient as pointer arithmetic
- Using subscripts requires using two variables: array name and index
- Compilers usually do not do extensive optimization by default
- Example: `gcc -O3`

Mergesort Revisited

- Let us look at a Mergesort algorithm implemented using pointer arithmetic
- Fill-in-the-blanks code available at:
  `~prof2132/public/mergesort2.c`