Computer Programming Lab 7

2023/11/21 Tim Chen

Contents

`const` Qualifier with Pointers

const Qualifier with Pointers

`const int a` - `a` is a constant integer.

`int const a` - `a` is a constant integer.

`const int *a` - `a` is a pointer, pointing to constant integer.

`int const *a` - `*a` is a constant integer.

`int * const a` - `a` is a constant pointer, pointing to integer.

const Qualifier with Pointers (cont.)

`const int a`

1. The type of `a` is `int`
2. `const` is applied to `int` which make this integer a constant.

const Qualifier with Pointers (cont.)

```
int const a
```

1. The type of `a` is `int`
2. `const` is applied to `a` which make this integer a constant.

const Qualifier with Pointers (cont.)

`const int *a`

1. The type of `a` is `int *`
2. `const` is applied to `int` which make this integer a constant.
3. Target integer is a constant, can't modify the integer.

```
*a = 5;         // error
a = &new_addr // allowed
```

const Qualifier with Pointers (cont.)

`int const *a`

1. The type of `a` is `int *`.
2. The type of `*a` is `int`.
3. `const` is applied to `*a` which make this integer a constant.

```
*a = 5;         // error
a = &new_addr // allowed
```

const Qualifier with Pointers (cont.)

`int * const a`

1. The type of `a` is `int *`.
2. `const` is applied to `a` which make this pointer to integer constant.
3. `a` is a constant integer pointer, can't modify the address.

```
*a = 5;          // allowed
a = &new_addr; // error
```

const Qualifier with Pointers (cont.)

`const int * const a`

1. The type of `a` is `int *`.
2. `const` is applied to `a` which make this pointer to integer constant.
3. `const` is applied to `int` which make this integer a constant.
4. `a` is a constant integer pointer, can't modify the address.
5. Target integer is a constant, can't modify the integer.

```
*a = 5;          // error
a = &new_addr; // error
```

Function

Function

```
void bubbleSort( int * const array, const int size );
```

Function (cont.)

```
void bubbleSort( int * const array, const int size );
```

1. `array` is a `int *`
2. `const` is applied to `array`, so this pointer can't be modified.
3. The value pointed by `array` is motifiable.

Quick Sort

Quick Sort

```
void qsort(
  void *base, size_t nitems, size_t size,
        int (*compar)(const void *, const void*)
)
```

Quick Sort (cont.)

```
int compar(const void * a, const void * b);
```

1. `a` is a pointer to `void`, so whatever `a` is pointing to is a constant.
2. Changing the type - `*(const int *)a`. ⟶ `a` has changed from `const void *` to `const int *` and `*a` is a constant integer.

Quick Sort (cont.)

```
int compar(const void * a, const void * b) {
  int l = *(const int *)a; // a is `const int *` now
  int r = *(const int *)b; // r is *b
  /* function return type
   * negative, if first < second
   * 0,        if first equal second
   * positive, if first > second
   */
}
```

Quick Sort (cont.)

```
int compar(const void * a, const void * b) {
  int l = *(const int *)a; // a is `const int *` now
  int r = *(const int *)b; // r is *b
  if (l - r > 0) return 1;
  if (l - r < 0) return -1;
  return 0;
}
```

Quick Sort (cont.)

```
int compar(const void * a, const void * b) {
  int l = *(const int *)a; // a is `const int *` now
  int r = *(const int *)b; // r is *b
  return l - r;
}
```

Quick Sort (cont.)

```
int compar(const void * a, const void * b) {
  return *(const int *)a - *(const int *)b;
}
```

Array of Pointer

Array of Pointer

```
const char *suit[4] = { "Hearts", "Diamonds", "Club",
"Spades" };
```

Array of Pointer (cont.)

```
const char *suit[4] = { "Hearts", "Diamonds", "Club",
"Spades" };
```

1. `suit` is an array.
2. `suit` is an array of `char *`.
3. `const` is applied to `char` so that characters pointed by `suit` i.e. `*suit` can't be modified.

```
char a = 'a';
*suit[0] = a; // error
```

Array of Pointer (cont.)

```
const char *suit[4] = { "Hearts", "Diamonds", "Club",
"Spades" };
printf("%c\n", *suit[0]); // H
printf("%c\n", *(suit[1] + 3)); // m
```

Function Pointer

Function Pointer

```
void bubble(int(*compare)(int, int)) {
  if ( (*compare)(a, b) ) return true;
}
```

or

```
void bubble(int(*compare)(int, int)) {
  if ( compare(a, b) ) return true;
}
```

Function Pointer (cont.)

```
typedef int (*compare)(int, int);
```

```
int bubble(compare cmp) {
  if ( (*cmp)(a, b) ) return true;
}
```

or

```
int bubble(compare cmp) {
  if ( cmp(a, b) ) return true;
}
```

Array of Function Pointer

Array of Function Pointer

```
int func1(int a) { return a + 1; }
int func2(int a) { return a - 1; }
int func3(int a) { return a; }
```

Array of Function Pointer (cont.)

```c
int main(void) {
  int (*f[3])(int) = { func1, func2, func3 };
}
```

Array of Function Pointer (cont.)

```
int main(void) {
  int (*f[3])(int) = { func1, func2, func3 };
}
```

1. Type of `f` - `int (*[3])(int)`.
2. To execute - `(*f[choice])(int);` or `f[choice](int)`

Array of Function Pointer (cont.)

```c
int main(void) {
  int (*f[3])(int) = { func1, func2, func3 };

  const int response1 = (*f[0])(4); // response1 = 5
  const int response2 = f[2](3);    // response2 = 3
}
```

Appendix A

typedef

typdef <existing_type> <alias>

```
/* Declares WHOLE to be a synonym for int */
typedef int WHOLE;
```

For example, `WHOLE` could now be used in a variable declaration such as `WHOLE i;` or `const WHOLE i;`. However, the declaration `long WHOLE i;` would be illegal.

`typedef` for structure

```
typedef struct club {
  char name[30];
  int size, year;
} GROUP;
```

This declare `GROUP` as a structure type with three members. Since structure tag `club` is also specified, either `GROUP` or structure tag can be used in declarations.

typedef for structure (cont.)

```
struct club {
  char name[30];
  int size, year;
}
typedef struct club GROUP;
```

```
struct club my_club;
GROUP my_club;
```

`typedef` for pointer

```
typedef struct club {
  char name[30];
  int size, year;
} GROUP;
typedef GROUP *PG;
```

The type `PG` is declared as a pointer to the `GROUP` type, which in turn is defined as a structure type.

Appendix B

Some Links

- Neovim - https://neovim.io/
- My dotfiles - https://github.com/gnitoahc/.dotfiles
- My email - 110703038@nccu.edu.tw

Thank you