FREE CHEAT SHEET

NUS CS1010 Practical Exam Cheat Sheet

One page of strategy, one page of C reference. The patterns that actually show up in the practical, written by a working software engineer who's tutored a lot of CS1010 students through it.

THE EXAM, DECODED

- Two hours, closed network, four problems of escalating difficulty.
- Pattern: 1 easy (compile + submit check), 1 medium (arrays / strings), 1 hard (pointers / recursion), sometimes 1 bonus.
- Marks come from **passing the provided test cases**, not from elegant code.

THE 5-STEP SURVIVAL PLAN

1. **Read every problem first, 5 minutes total.** The hardest problem might be 80% similar to a tutorial you already solved.
2. **Solve in increasing difficulty.** Bank the easy points before risking the hard ones.
3. **Test on the provided cases before moving on.** A working solution beats an "almost there" cleverer one.
4. **If stuck for 20 minutes, switch.** Come back to it, often you spot the bug fresh.
5. **Last 10 minutes: re-test.** Some students fix one thing and break another in the final stretch.

POINTERS, THE PARKING-SPOT MODEL

A pointer is a **label on a parking spot**. The variable is the car parked in the spot. `*p` means "look at the car in the spot p points to". `&x` means "the label of x's spot".

```c
int car = 42;
int *label = &car;        // label points to car's spot
*label = 99;                // replace the car at that spot
printf("%d\n", car);     // 99
```

`int *p` in declarations means "p is a pointer to int". `*p = 99` in code means "the int at p". Different things, same symbol. Read every `*` in context.

FORMAT STRINGS (PRINTF / SCANF), THE ONES YOU'LL NEED

printf

```c
printf("%d", n);        // int
printf("%ld", big);      // long
printf("%f", x);        // double
printf("%.2f", x);      // 2 decimals
printf("%c", ch);       // char
printf("%s", str);      // C string
```

scanf, take an address

```c
int n;
scanf("%d", &n);

double x;
scanf("%lf", &x);      // %lf for double
```

```c
printf("%5d", n);       // width 5
printf("\n");           // newline
```

```c
char name[100];
scanf("%99s", name);    // %s does not need &
                        // limit prevents overflow
```

RECURSION, THE TEMPLATE

```c
int factorial(int n) {
    if (n <= 1) return 1;       // base case, always first
    return n * factorial(n - 1);  // recursive case, smaller input
}
```

Every recursive function needs **a base case** and **each recursive call must shrink the input**. Miss either and you stack-overflow.

FIVE BUGS THAT COST MARKS

- **Off-by-one**: `for (i = 0; i <= n; i++)` visits index `n`, but a length-n array only has `0..n-1`. Use `<`.
- **Forgetting `&` in scanf** for non-string types. Compiler may not warn. Output looks like garbage.
- **Missing null terminator** when building strings manually. Always end with `'\0'`.
- **Integer overflow**: `int` max is ~2.1 billion. For factorials, big sums, use `long long`.
- **Returning a pointer to a local**: the local goes out of scope, you have a dangling pointer.

THE DAY OF

- Wake up alert, normal breakfast, no surprises.
- Skim this sheet once before going in.
- Bring water and your IC. Pencils don't matter, you're typing.
- If something compiles but is wrong, comment it out and submit a partial. Marks for compiling.