Algorithms and Data Structures

Common Errors in C Code

Stefano Quer
Department of Control and Computer Engineering
Politecnico di Torino

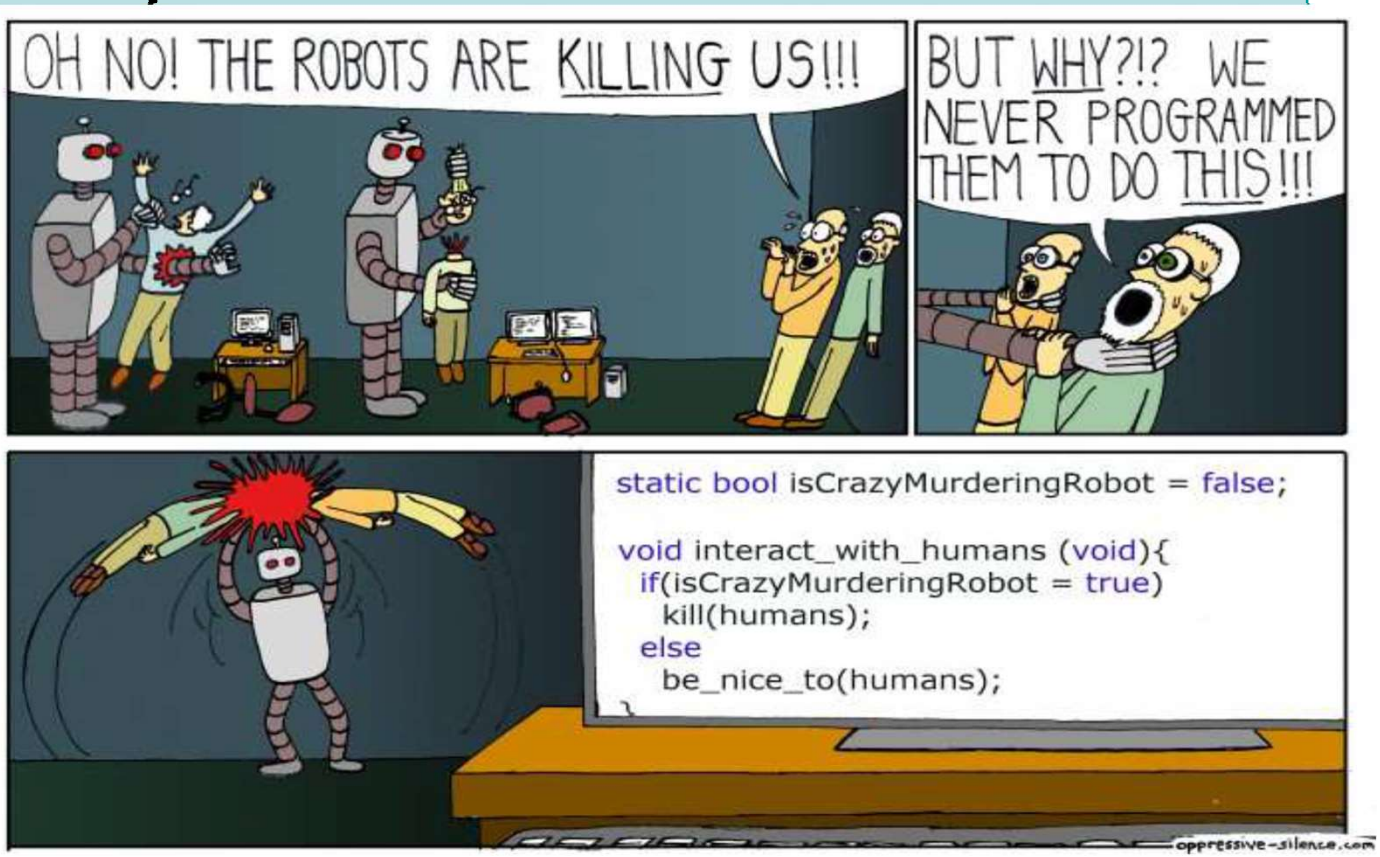
Test 1 ...

```
1. int i=3, j, k=i, s;
2. ..
3. s=i+j; p=i*i;
4. ...
5. if (i=3) { ... }
6. ...
7. if (k==i);
8.   i++;
9. ...
```

- Which are the errors included in the code snipped above?

...

```
1. int i=3, j, k=i, s;
2. ..
3. s=i+j; p=i*i;
4. ...
5. if (i=3) {      }
6. ...
7. if (k==i);
8.   i++;
9. ...
```



...

```
1. int i=3, j, k=i, s;
2. ..
3. s=i+j; p=i*i;
4. ...
5. if (i=3) {      }
6. ...
7. if (k==i);
8.   i++;
9. ...
```

... Test 1

```
1. int i=3, j, k=i, s;
2. ..
3. s=i+j; p=i*i;          // No, Uninit j, Undef p
4. ...
5. if (i=3) { ... }       // No, = → ==
6. ...
7. if (k==i);             // No, ; → no ;
8.   i++;
9. ...
```

- Be precise
  - The compiler does not reveal all errors !

Test 2

```
int main () {
  int n;
  scanf ("%d", &n);
  printf ("%d", n);
  fprintf (stdout, "%d", n);
  return 1;
}
```

stdout ←→ stderr

- Questions
  - Why there is an & in scanf and not in printf?
  - Which is the difference between prinf and fprintf?

Test 3

```
int main () {
  char s[10];
  scanf ("%s", &s);
  printf ("%s", s);
  return 1;
}
```

- Is the code snipped correct?

Test 4 ...

```
int x = 5, i;
float y = 2, f;

i = x/y;
f = x/y;

printf ("%f %f\n", (float) i, f);
```

- What does printf print?

... Test 4

```
int x = 5, i;
float y = 2, f;

i = x/y;
f = x/y;

printf ("%f %f\n", (float) i, f);
```

`j = x%y;`
is the remainder, i.e., 1

- Pay attention to integer divisions
  - $\frac{N}{D} = q + \frac{R}{D}$
  - $\frac{5}{2} = 2 + \frac{1}{5}$ **printf** prints 2.0 2.5

N = Numerator
D = Denominator
R = Remainder
Q = Quotient

Possibly, use
`f=((float)x)/(foat)y);`

Test 5 ...

```
int value;
do {
   ...
   value=10;
} while(!(value==10) || !(value==20))
```

- Even though value is 10 the program loops
  - Why?

...

```
int value;
do {
   ...
   value=10;
} while(!(value==10) || !(value==20))
```

- Do not misuse Boolean operators
  - The condition given above is a tautology
  - It is always true, as value can't be both values at once

... Test 5

```
int value;
do {
   ...
   value=10;
} while(!(value==10) && !(value==20))
```

- It is necessary to use &&
  - Which reads much more nicely: "if value is not equal to 10 and value is not equal to 20", which means if value is some number other than ten or

Test 6 ...

```
1. int i, n;
2. int v1[5], v2[n];
3. ..
4. for (i=1; i<=5; i++)
5.   scanf ("%d", &v1[j]);
```

- Question
  - Is the code snipped correct?

... Test 6

Possibly, use
`#define N 10`
`int v2[N];`

```
1. int i, n;
2. int v1[5], v2[n];            // No, n not const
3. ..
4. for (i=1; i<=5; i++)         // No, i=0, i<5
5.   scanf ("%d", &v1[j]);
```

- Arrays in C
  - Can, at least for now, be defined only of constant size
  - Always start at index 0; thus not not overstep array boundaries

Test 7 ...

```
1. char c = ‘a’;
2. char c = 21;
3. char c = 1234;
4. char ch = 'A’;
5. char ch = "A";
6. const char *st = "A";
7. const char *st = 'A';
```

- Q
  - Which ones among the previous definitions are correct?

... Test 7

```
1. char c = 'a';                    // Correct
2. char c = 21;                     // Correct
3. char c = 1234;                   // No, OW
4. char ch = 'A';                   // Correct
5. char ch = "A";                   // No, String
6. const char *st = "A";            // Correct
7. const char *st = 'A';            // No, Char
```

- C considers character and string constants as very different things
  - Character constants are enclosed in single quotes
  - String constants are enclosed in double quotes
    - String constants act as a pointer to the actually string

Test 8 ...

```
char st1[] = "abc";
char st2[3] = "abc";
if (st1 == st2)
  printf("Yes");
else
  printf("No");
```

- Question
  - Is the code snipped correct?

...

```
char st1[] = "abc";
char st2[3] = "abc";                    // No, must include '\0'
if (st1 == st2)
  printf("Yes");
else
  printf("No");
```

- A C string must have a null terminator at the end of the meaningful data in the string

`char str[]={'a','b','c'};`
is **not** a string

...

```
char st1[] = "abc";
char st2[3] = "abc";              // No, must include '\0'
if (st1 == st2)                   // No, must use strcmp
  printf("Yes");
else
  printf("No");
```

- A C string must have a null terminator at the end of the meaningful data in the string
  - To manipulate strings use the proper C library function

```
if (strcmp(st1,st2)==0))
```

... Test 8

```
char st1[] = "abc";
char st2[3] = "abc";              // No, must include '\0'
if (st1 == st2)                   // No, must use strcmp
  printf("Yes");
else
  printf("No");
```

- Pay attention to the switch statement

```
switch (str) {
  case "123": ...
  case "ab": ...
}
```

Constants must be countable

Test 9 ...

```
int main () {
  menu();
  ...
}
void menu() { ... }
```

- Question
  - Is the code snipped

... Test 9

```
int main () {
  menu();
  ...
}
void menu() { ... }
```

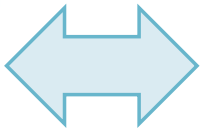
More details on that on the ADT and modularity unit

- The compiler doesn't know what menu stands for until you have told it
  - You cannot wait until after using it to tell it that there's a function
  - Always remember to put either a **prototype** for the function or the entire definition of the function **above** the first time you use the function

Test 10 ...

```
#define SQUARE(x) x*x
#define MAX(x,y)  ((x>y)?x:y)

z = SQUARE(1+2);
z = MAX(2,5);
```



```
if(x>y)
  z=x;
else
  z=y;
```

```
z=(x>y)?x:y;
```

- Do not forget the use of macros
  - Macros are simple string replacements
  - So, they will work with preprocessing tokens
- Use them correctly
  - What does the code print?

... Test 10

```
#define SQUARE(x) x*x
#define MAX(x,y)  ((x>y)?x:y)

z = SQUARE(1+2);
z = MAX(2,5);
```

- Macro SQUARE computes
  - 1+2*1+2=5, not 3*3=9
  - Instead, use

```
#define SQUARE(x) ((x)*(x))
```

Test 11 ...

```
FILE *fp = fopen("test.txt", "r");
char line[100];
while (!feof(fp)) {
  fgets(line, sizeof(line), fp);
  ...
}
fclose(fp);
```

- Question
  - Is the code snipped correct?

...

```
FILE *fp = fopen("test.txt", "r");
char line[100];
while (!feof(fp)) {
  fgets(line, sizeof(line), fp);
  ...
}
fclose(fp);
```

Mainly, it comes from Pascal (where it checks the result of the next instruction)

- There is a wide spread misunderstanding of how C's feof function works
  - The function returns true if the last function failed
  - The program will print the last line of the input file **twice**

... Test 11

- Use the following code instead

```
while (1) {
  fgets(line, sizeof(line), fp);
  if (feof(fp))
    break;
  ...
}
fclose(fp);
```

```
while (fgets(line, sizeof(line), fp) != NULL) {
...
}
fclose(fp);
```

Or

```
while (fscanf (fp, …) != EOF) {…}
```

Test 12 ...

```
char *st;
strcpy (st, "abc");
```

- Question
  - Is the code snipped correct?

...

```
char *st;                          // st points to nothing
strcpy (st, "abc");
```

- Anytime you use a pointer, you should be able to answer the question
  - What variable does this point to?
  - If you can not answer this question, it is likely it doesn't point to any variable
  - This type of error will often result in a Segmentation fault/coredump error on UNIX/Linux or a general protection fault under Windows

... Test 12

```
char *st;                          // st points to nothing
strcpy (st, "abc");
```

- You need to use
  - Dynamic memory allocation
  - Function **strdup** (which is the same)

First topic of "Algorithm and Data Structures"

```
char *st;
st = strdup ("abc");
```