C for C++ programmers

Prof. Dr.–Ing. Damian Weber

Abstract

What the C++ programmer must know when using a C compiler.

1 Input/Output

1.1 Standard Output

1.1.1 C++

```
#include <stream.h>

int a;
double x;
char *s="xyz";

// print string, integer and floating point
cout << "s=" << s << " a=" << a << " x=" x << endl;
```

1.1.2 C

```
int a;
double x;
char *s="xyz";

/* print string, integer and floating point */
printf("s=%s a=%d x=%lf\n",s,a,x);
```

More info: `man printf` on a UNIX system

1.2 Standard Input

1.2.1 C++

```
#include <stream.h>

int a;
double x;
char s[20];

// input string, integer and floating point
cout << "Input s: " << flush;
cin >> s;
cout << "Input a: " << flush;
cin >> a;
cout << "Input x: " << flush;
cin >> x;
```

Note: `flush` forces the output buffer written to the terminal

1.2.2 C

```
#include <stdio.h>

int a;
double x;
char s[20];

/* print string, integer and floating point */
printf("s=);
fflush(stdout);
scanf("%s",s);

printf("a=);
fflush(stdout);
scanf("%d",&a);

printf("x=);
fflush(stdout);
scanf("%lf",&x);
```

Note: `fflush(stdout)` forces the output buffer written to the terminal

More info: `man scanf` on a UNIX system

1.3 File Output

1.3.1 C++

```
#include <fstream.h>

int a;
double x;
char *s="xyz";
ofstream fout;

fout.open("datafile");
if (!fout) return; // could not open file
```

```
// write string, integer and floating point
fout << s << endl;
fout << a << endl;
fout << x << endl;

// ofstream destructor calls close
```

1.3.2 C

```
#include <stdio.h>

int a;
double x;
char *s="xyz";
FILE *f;

f=fopen("datafile","w");
if (!f) return; /* could not open file */

/* write string, integer and floating point */
fprintf(f,"%s %d %lf",s,a,x);

fclose(f);
```

1.4 File Input

1.4.1 C++

```
#include <fstream.h>

int a;
double x;
char s[20];
ifstream fin;

fin.open("datafile");
```

```
if (!fin) return; // could not open file

// read string, integer and floating point
fin >> s;
fin >> a;
fin >> x;

// ifstream destructor calls close
```

1.4.2 C

```
#include <stdio.h>

int a;
double x;
char s[20];
FILE *f;

f=fopen("datafile","r");
if (!f) return; /* could not open file */

/* read string, integer and floating point */
fscanf(f,"%s %d %lf",s,&a,&x);

fclose(f);
```

Note:

- for checking on end–of–file, there exists a function called `int feof(FILE *)`.
- for reading complete lines of a file, there exists a function called `char *fgets(char *, int, FILE *)` which returns the NULL pointer on error or end–of–file.

2 Memory Allocation

main motivation: create arbitrarily sized arrays

2.1 C++

```
int *a;  // becomes an int array
char *s;  // becomes an char array (string)

a=new int[100]; // now a[0]...a[99] accessible
s=new char[20]; // now s[0]...s[19] => string of size 19 + ending \0

...

delete [] a; // deallocate memory
delete [] s; // deallocate memory
```

2.2 C

```
int *a;  /* becomes an int array */
char *s;  /* becomes an char array (string) */

a=malloc(100*sizeof(int)); /* now a[0]...a[99] accessible */
s=malloc(20*sizeof(char)); /* => string of size 19 + ending \0 */

...

free(a); /* deallocate memory */
free(s); /* deallocate memory */
```

3 String Handling

Strings are arrays of `char`.

3.1 Initialization

Make sure you have enough memory allocated for your string. Maybe you should read section 2 again.

3.2 Conversions Integer ⟷ String

3.2.1 Integer ⟶ String

We print the `int` (or `long`) into a `char` array:

```
...
int i;
long l;
char str[13];

i=12345;
sprintf(str,"%d",i);
sprintf(str,"%ld",l);
...
```

3.2.2 String ⟶ Integer

We read the `int` (or `long`) from a `char` array:

```
...
int i;
char str[13];

strcpy(str,"12345");
sscanf(str,"%d",&i);
...
```

Both functions require the inclusion of `stdio.h`.

Alternatively, we can use `atoi()`, this requires `stdlib.h`:

```
...
int i;
char str[13];

strcpy(str,"12345");
```

```
        i=atoi(str);
        ...
```

4 Miscellaneous

4.1 Call–by–Pointer

In C, there is no call–by–reference, only call–by–pointer.

4.1.1 C++

```
class date
{
   public: // only for demonstration
           // please don't use public member vars
   int day,month,year;
}

int some_function(date &d)
{
    cout << d.day << "." << d.month << "." << d.year;
}

void call_some_funtion()
{
   struct date dd;

   dd.day=12;
   dd.month=10;
   dd.year=2003;

   some_function(dd);
}
```

4.1.2 C

```
struct date
{
   int day,month,year;
}

int some_function(struct date *dp)
{
   printf("%d.%d.%d",dp->day,dp->month,dp->year);
}

void call_some_funtion()
{
   struct date dd;

   dd.day=12;
   dd.month=10;
   dd.year=2003;

   some_function(&dd);
}
```

4.2 Declaration

Declarations must happen strictly before the first statement in a function.

The following is not possible in C:

```
for (int i=0;i<=10;i++)
    printf("%d\n",i);
```

Instead, use:

```
int i;

for (i=0;i<=10;i++)
    printf("%d\n",i);
```