Introduction to C

Programming Workshop in C (67316)
Fall 2017
Lecture 4
2.11.2017

Pointers are variables that store the **address of other variables**

- **Declaration**

  `<type> *p;   (e.g. int *p;)`

  p points to object of type `<type>`

- **Pointer → value (de-reference)**

  `*p` refers to the object p points to

  `(e.g. *p = x;  y = *p;)`

- **Value → pointer**

  `&x` - the address of x `(e.g.  p = &y;)`

Pointers & Arrays

Array name can **sometimes** be treated as the address of the first member.

```c
#include <stdio.h>
int main()
{
  char arr[] = "don't panic\n";
  char* ptr = arr;
  printf("%c %c\n", arr[4], ptr[4]);
  printf("%c %c\n", *(arr+2), *(ptr+2));
  return 0;
}
```

Pointer arithmetic and array indexing are equivalent.

Pointers & Arrays

Arrays passed to functions are converted to pointers

```
int sum (int arr[])
{
   int i, sum = 0;
   for (i=0; i<sizeof(arr)/sizeof(arr[0]); ++i)
   {
      sum += arr[i];
   }
   return sum;
}
```

Logical error:
```
sizeof (arr) ==
sizeof (int*) ==
sizeof (void*)
```

Pointers & Arrays - is there a difference?

Arrays can’t be manipulated like pointers

```
#include <stdio.h>
int main() {
   int i;
   char array[] = "don't panic";
   char* ptr = array;

   /* array traversal */
   for (i = 0; i < sizeof(array); ++i)
        printf("%c ", array[i]);

   printf("\n");

   /* pointer traversal */
   for (; *ptr; ++ptr)
        printf("%c ", *ptr);

   return 0;
}
```

An array has to be indexed with another variable

void *

`void *p` defines a pointer to <u>undetermined type</u>

```
int j;
int *p = &j;
void* q = p;  // no cast needed
p = (int*)q ; // cast is needed
```

All pointers can be casted one to the other, it may be useful sometimes, but beware…

void *

- No pointer arithmetic is defined for void*
  (gcc has an extension, treating the size of a void as 1)
- We cannot access the content of the pointer – dereferencing is not allowed

```
int j;
void *p = &j;
int k = *p;           // illegal
int k = (int)*p ; // still illegal
int k = *(int*)p; // legal
```

Pointers to pointers to pointers to...

Pointer is a variable type, so we can create a pointer to pointer.

```
int main()
{
   int n = 17;
   int *p = &n;
   int **p2 = &p;
   printf("the address of p2 is %p \n", &p2 );
   printf("the address of p is %p \n",   p2 );
   printf("the address of n is %p \n",  *p2 );
   printf("the value of n is %d \n",   **p2 );
   return 0;
}
```

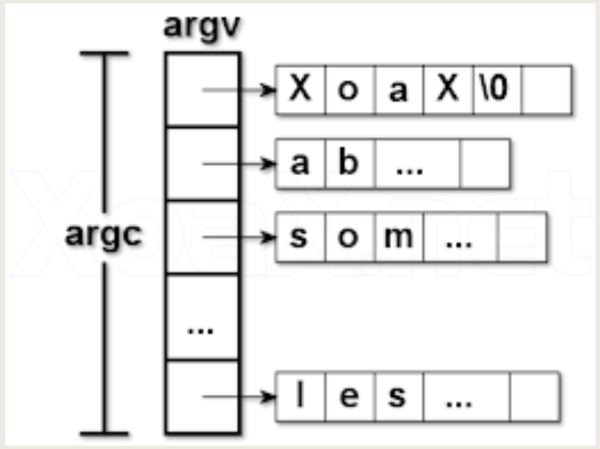
Passing arguments to a program with argc and argv

argc

- stands for "**arg**ument **c**ount"
- contains the number of arguments passed to the program

argv

- stands for "**arg**ument **v**ector"
- array of strings



```
> myprog 1 2 3
```

```
argc = 4 (program name is the first)
argv[0] => "myprog"
argv[1] => "1"
argv[2] => "2"
argv[3] => "3"
```

Passing arguments to a program with argc and argv

- it is a good practice to print program arguments at the beginning of the program
- when the number of arguments is not what you expect, it is a good practice to print program usage

```c
int main(int argc, char *argv[])
{
   for(int i=0; i<argc; i++)
   {
     printf("%s ", argv[i]);
   }

   if(argc < 2) // no arguments given
   {
     printf("Usage: myprog <num1> <num2>\n");
   }
}
```