Introduction to pointers (and strings, arrays revisited)

Strings

- string is a character array whose last character is '\0'
- string is not a data type in C
- and hence comparing two strings, or assigning a string has to done explicitly through functions
  - Eg: char str[ ] = "hello";

```
char name[ ] = "world";
if (name == str) /* this compares just starting addresses*/
{
        …;
}
```

Strings

- Most common functions for strings are defined in a standard library with header file *string.h*
- For example:
  - int strlen(char s[ ])
    {
      int j = 0;
      while(s[j] != '\0') ++j;
      return j;
    }

Addresses and Pointers (a brief)

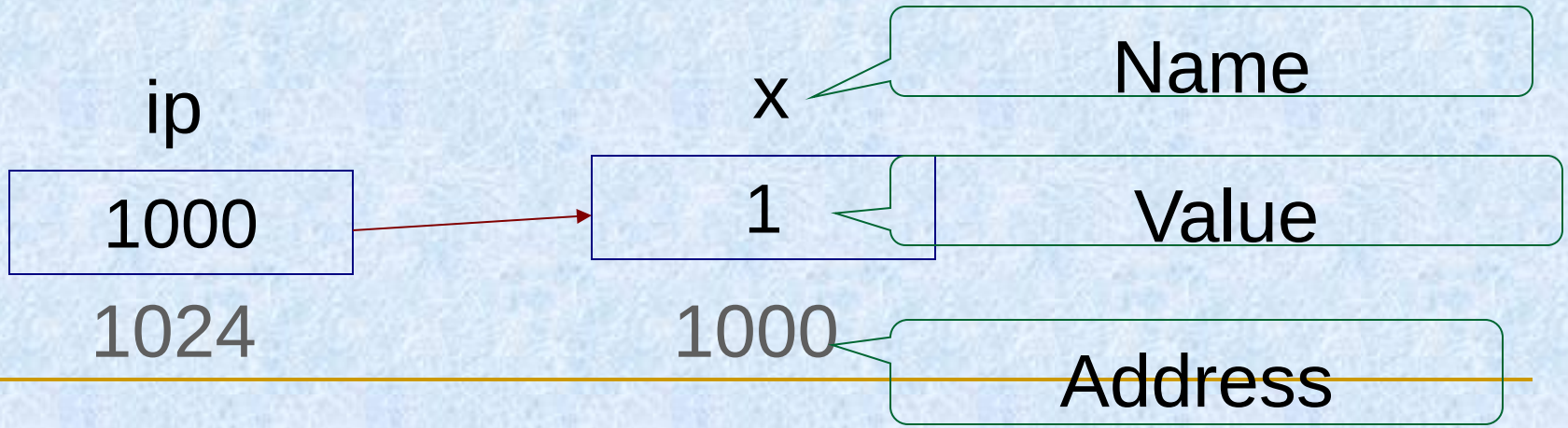
- int j = 5;
- The unary operator & gives the address of its operand.
- Hence, &j is the address of j
- & cannot be applied to expressions, constants, or register variables.
- Let &j = 1000.
- Even by knowing this 1000, if you do not know that the value at the address is an integer, you can not retrieve 5.
- So, along with address, we need the type of the value located at the address.

Pointers

- Pointer is a variable that contains the address of a variable.
- int *p; /*declaration of a pointer*/
- p is an integer pointer.
- That is, value of p is an address. And at that address an integer is stored.
- The unary operator ***** is the *indirection* or *dereferencing* operator; when applied to a pointer, it accesses the object that is pointed by the pointer.

Pointers, an example

```
int x = 1, y = 2, z[10];
int *ip;          /* ip is a pointer to int */
ip = &x;          /* ip now points to x */
y = *ip;          /* y is now 1 */
*ip = 0;          /* x is now 0 */
ip = &z[0];       /* ip now points to z[0] */
```



Pointers

```
void swap (int *, int *);
main( )
{
   int j = 10, k = 20;
   swap(&j, &k);
   printf("%d  %d", j, k);
}
void swap(int *a, int *b)
{
  int t;
  t = *a, *a = *b,  *b = t;
}
```

- This is the correct way of writing a swap function.

Pointers and Arrays

- Any operation that can be achieved by array subscripting can also be done with pointers.
- int a[10] = {1,2,3,4,5};
  int *ip;

  ip = a;
  ip[0] = 10; /* is same as a[0] = 10 */
  (ip + 5) = 20; / same as ip[5] = 20 */
- When you say, ip + 5 it is indeed
  ip + 5*sizeof(int)
  this is called pointer (address) arithmetic.

strlen with pointers

```
int strlen(char s[ ])
{
  int j = 0;
  while(s[j] != '\0')
          ++j;
  return j;
}
```

```
int strlen(char *p)
{
  int j = 0;
  while(*p != '\0'){
          ++p;
          ++j;
  }
  return j;
}
```

- Both are correct
- But there is a subtle difference between them

String constants

```
#include<string.h>
main( )
{
   int j, k;
   char str[ ] = “hello world”;
   char *p;
   p = str;    /* Ok  */
   p = “Hi is this right”;
   j = strlen(p);
   k = strlen(“how are you”);
}
```

- String constant
- Just like 15 is a numeric constant
- This is OK
- This is also OK

String Assignment

- char str[4] = “cat”;
- char name[4];
- char *ptr;
- name = str ; /* Is this right ? */
- /* name is constant, hence error */
- ptr = str; /* This is OK */
- ptr[1] = ‘o’; /* this results in str[1] = ‘o’ */
- /* to copy all elements of str into name */
- strcpy(name, str); /* available in string.h */
- /* element by element str array is copied into name array */
- name[1] = ‘o’; /* this does not mean str[1] = ‘o’ */
- /* name[ ] and str[ ] are two different arrays */

strcpy

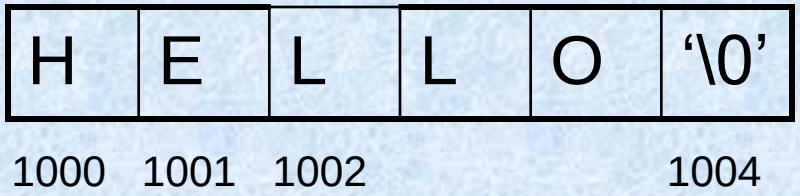
```
void strcpy(char *s, char *t)
{
        *s = *t;
        if(*t == '\0') return;
        while(*t != 0) {
                s++,  t++;
                *s = *t;

        }
 }
```

A closer look

- char amessage[ ] = “now is the time”;
- char *pmessage = “now is the time”;
- amessage is an array. It is initialized.
- Individual characters within the array may be changed but amessage will always refers to the same memory location.
- But pmessage is a pointer, initialized to point to a string constant.
- Doing like, pmessage = “new string”; is OK
- But you cannot do like

  amessage = “new string”;

String constant



- char *p = "HELLO";
- Some where in memory, "HELLO" is stored.

- That memories starting address is assigned to p. So, p has value 1000
- Hence, "HELLO" has value 1000
- printf("%s", "ravi");

  /*this is perfectly correct */

strcpy revisited

- void strcpy(char *s, char *t)

```
{
        int j=0;
        while( (s[j] = t[j] )  !=  '\0') j++;
}
```

- void strcpy(char *s, char *t)

```
{
        while( (*s = *t )  !=  '\0'){
                s++;
                t++;
        }
}
```

- void strcpy(char *s, char *t)

```
{
        while( *s++ = *t++ );
}
```