CS 3411 Systems Programming

Department of Computer Science
Michigan Technological University

C vs. C++ (cont.)

Examples of Pointer Use: Strings in C

- There is no string data type in C.
- Instead, a string is assumed to be a sequence of `char` terminated by a zero byte.
- A `char *` is generally used as a string; just a pointer to the first char in the zero-terminated sequence of chars.
- Careful when *declaring* a string:

```
char *STR; /* Only memory allocated is to pointer variable */
char str[20]; /* 20 bytes allocated to hold contents of string *
```

String Codes Example I

Some examples of string functions:

```
#include <stdio.h>
int mystrlen(s)
char *s;
{
  char *p;
  p = s;
  while (*p) p++;
  return p-s;
}

strcpyv1(t, s)
char *s, *t;
{
  while ((*t = *s) != '\0') {
    s++;
    t++;
  }
}
```

String Codes Example II

```c
strcpyv2(t, s)
char *s, *t;
{
  while ((*t++ = *s++) != '\0');
}

strcpyv3(t, s)
char *s, *t;
{
  while (*t++ = *s++);
}

main() {
  char test_str[] = "Hello␣World!";
  char copy_to_str[20];

  printf("Original␣string:␣%s\n",test_str);
  printf("Length␣of␣string:␣%d\n", mystrlen(test_str));
  strcpyv3(copy_to_str,test_str);
  printf("Copied␣string:␣%s\n",copy_to_str);
  printf("Length␣of␣string:␣%d\n", mystrlen(copy_to_str));
}
```

Manual Pages

- Usually the most accurate source of information for the system you're working on
- Accessed by the 'man' command from the terminal, followed by section number, followed by the item you want information on
- Sections vary from system to system. You can see this by using the command 'man man'. Commonly, the sections are:
  1. User commands
  2. System calls
  3. Library routines
  4. Devices
  5. File formats
  6. Games
  7. Misc.
  8. System Administration
- The 'info' command is another option for GNU Software

C Standard I/O

- Different from C++
- Manual pages available for specific functions:
  - man 3 stdio (An overview)
  - man 3 printf (Formatted output)
  - man 3 scanf (Formatted input)
  - man 3 getc (Character-based input macros)
  - man 3 putc (Character-based output macros)
- Default I/O Streams: stdin, stdout, stderr
- Anything you open with the fopen function is also a stream.
- All streams are of the `(FILE *)` data type.

Output in C++

- C++ iostream methods « and » automatically format.

```
#include <iostream>
using namespace std;
main() {
  float x;
  int y;
  char *str;
  x = 3.1;
  y = −20;
  str = "Characters";
  cout << x << "␣" << y << "␣" << str << "\n";
}
```

Output in C

- stdio requires a string which defines a format to be used

```
#include <stdio.h>

main() {
  float x;
  int y;
  char *str;
  x = 3.1;
  y = -20;
  str = "Characters";
  printf("%.2f␣%d␣%s\n", x, y, str);
}
```

Input in C++

- C++ iostream style input:

```
#include <iostream>
using namespace std;
main() {
  double sum = 0;
  int val, num = 0;
  while (cin >> val) {
    num++;
    sum += (double) val;
  }

  cout << "Mean␣is␣" << sum/(double)num << "\n";
}
```

Input in C

- In C, we need to pass a pointer argument to scanf to get back values

```
#include <stdio.h>

main() {
  double sum = 0;
  int val, num = 0;
  while (scanf("%d", &val) == 1) {
    num++;
    sum += (double) val;
  }

  printf("Mean␣is␣%f\n", sum/(double)num);
}
```

Memory Allocation in C

- No new/delete in C!
- Memory allocation is done through `malloc`
- Freeing memory is done through `free`
- ’man 3 malloc’ for more details!

Malloc Example I

```c
/* bintree.c */
#include <malloc.h>
#define NILNODE (struct node *)0

struct node {
  char data;
  struct node *left, *right;
};

main() {
  struct node *gimme(), *n1, *n2, *n3, *n4, *n5, *n6, *n7;
  void inorder();

  n1 = gimme('a', NILNODE, NILNODE);
  n2 = gimme('b', NILNODE, NILNODE);
  n3 = gimme('c', n1, n2);
  n4 = gimme('d', NILNODE, NILNODE);
  n5 = gimme('e', n3, n4);
  n6 = gimme('f', NILNODE, NILNODE);
  n7 = gimme('g', n5, n6);
  inorder(n7);
  printf("\n");
}
```

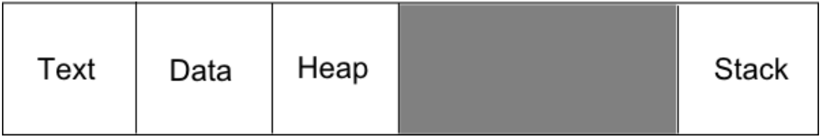
Malloc Example II

```
struct node *gimme(val, l, r)
char val;
struct node *l, *r;
{
  struct node *tmp;

  tmp = (struct node *) malloc(sizeof(struct node));
  tmp->data = val;
  tmp->left = l;
  tmp->right = r;
  return(tmp);
}

void inorder(r)
struct node *r;
{
  if (r != NILNODE) {
    inorder(r->left);
    printf("%c", r->data);
    inorder(r->right);
  }
}
```

A Brief Look at Program Execution



- Text is executable code (also some strings! Usually write-protected)
- Data is *global* data (both initialized and uninitialized)
- Heap is area from which dynamic allocations are made (malloc!)
- Stack is where function *activation records* pushed/popped.
  - Pushed (created) on stack when function invoked, removed on return
  - May contain: function parameters, function locals, return address, temporaries, saved state, control link, access link
- Usual to preallocate a block of storage for initial heap/stack

Problems to Avoid

- It is always important to keep system programs as bug-free as possible
- Errant programs running in privileged mode can:
  - Access/modify system configuration files
  - Erase user data
  - Halt the system
  - And so on!

Buffer Overflow

- Writing beyond allocated array bounds

```
int getUserData() {
  char copy[60];
  ...
  /* User can input string of ANY length */
  gets(buf);
  ...
  /* Copies until string termination in buf */
  strcpy(copy,buf);
}

main() {
  ...
  char input[50];
  char *strPtr;
  ...
  getUserData();
  /* No string memory allocation for strPtr */
  strcpy(strPtr,input);
}
```

Memory Leak

- Losing access to allocated memory segment - We can't reclaim it!

```
int func()
{
  void *ptr;
  /* When function returns, value of ptr inaccessible */
  ptr = malloc(100);
}

main() {
  char *bptr;

  for (i=1;i<10;i++) {
    /* Previous ptr value overwritten each iteration */
    bptr = malloc(sizeof(char));
    *bptr = i;
  }
}
```

Dereference Invalid Pointer

```
...
int func(node *n) {
  if (n->value == 0) free(n);
  return (0);
}

main() {
  node *p,*q;
  p = malloc(sizeof(node));
  p->value = 10;
  printf("Node␣p␣value␣<%d>",p->value);

  func(p);
  /* p has already been freed */
  printf("After␣func␣p␣value␣<%d>\n", p->value);
  /* q was never initialized */
  printf("Node␣q␣value␣<%d>\n",q->value);
}
```