Pointers, command line arguments, Process control, and functions

Prof. (Dr.) K.R. Chowdhary, Director SETG
Email: kr.chowdhary@jietjodhpur.ac.in
webpage: http://www.krchowdhary.com

Jodhpur Institute of Engineering and Technology, SETG

October 5, 2015

Pointers

- Pointer is data that holds the address of another memory item
- A pointer itself can store the address of another pointer

1. `int var1,var2,*ptr;`
2. `*ptr = 1234;`
3. `var1 = *ptr;`
4. `var2 = 1235;`
5. `ptr = &var2;`

- The operator & can be legally used only for variables and array elements, but not for compound expressions and constants: `ptr = &a[5];` is valid, and `ptr = &(a+b);` is invalid.
- The operator * can only be applied to pointer variables and expressions. see prog ptr12.c

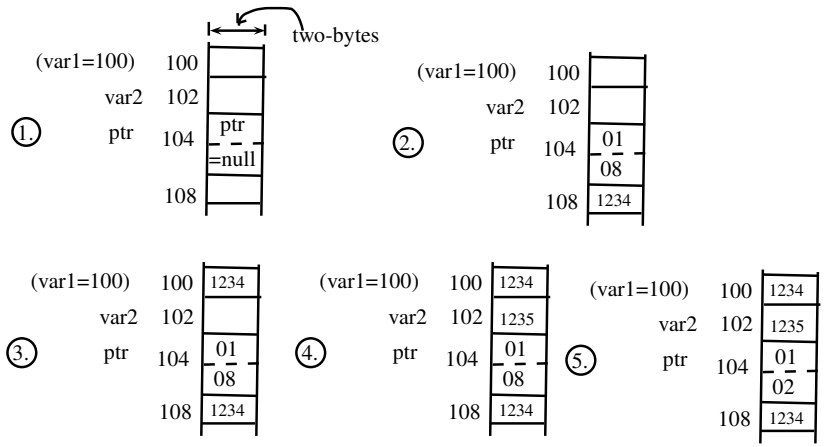


Figure 1: Pointer examples.

Pointers

```
 /* ptr12.c*/
#include <stdio.h>
int main(){
  int var1, var2, *ptr;
  *ptr = 1234;
  var1 = *ptr;
  var2 = 1235;
 printf("var1-m=%u\n", &var1);
 printf("var2-m=%u\n", &var2);
 printf("ptr-m=%u\n", ptr);
 printf("*ptr=%d\n", *ptr);
  ptr = &var2;
 printf("var2-val=%d\n", var2);
 printf("*ptr=%d\n", *ptr);
 printf("ptr-m=%d\n", ptr);
return 0;
}
```

Pointers

```
/* cptr1.c*/
#include <stdio.h>
int main(){
 int i=18, *ip , **ipp;
 ip = &i;
 ipp=&ip;
 printf("ip = %u, &ip = %u,
  ipp=%u \n", ip, &ip, ipp);
 printf("&i = %u, i = %d \n",
  &i, i);
 printf("hello\n");
 return 0;
}
```

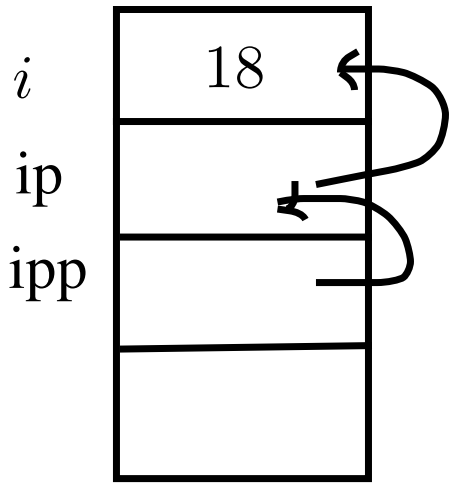


Figure 2: Double pointers.

Pointers

- A pointer variable may be assigned to a pointer variable of same type
- Pointer variable can be incremented or decremented
- difference between two pointer variable can be obtained by ptr1 - ptr2.

  Array pointer variables:
- `ptr` = `&arr[0]`;
- Its contents can be accessed by: `arr[0]` or `*ptr`
- The `arr[1]` can be accessed by `*(ptr + 1)`

Relation between arrays and Pointers

```
/* cptr2.c*/
#include <stdio.h>
char carr[4] = "ABC";
double darr[3] = {1.2, 3.4, 5.6};
int main(){
 char *cptr = &carr[0];
 double *dptr = &darr[0];
 for(; *cptr; cptr++, dptr++) {
    printf("*cptr: %c, cptr: %u ", *cptr, cptr);
    printf("*dptr: %g, dptr: %u\n", *dptr, dptr);
    ;}
 return 0;
}
```

Pointers

Representation or the array “arr” in memory:

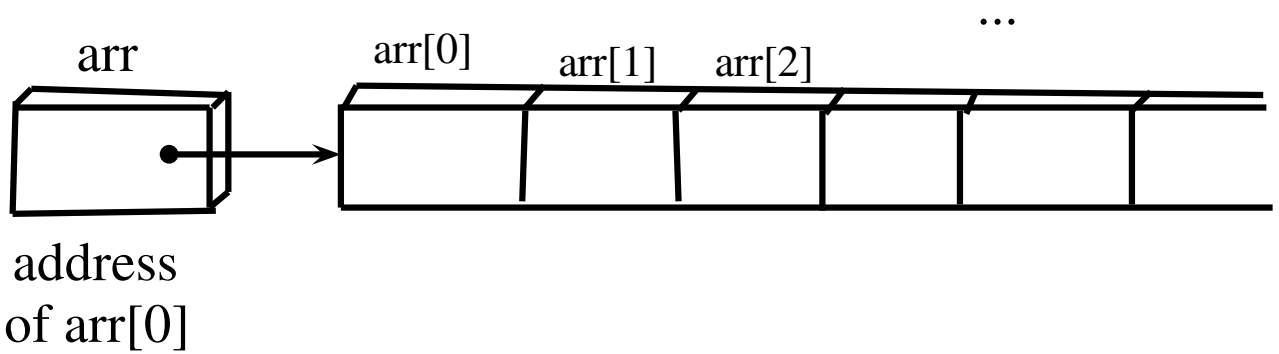


Figure 3: Pointers and arrays

Command Line arguments

- The C environment supports passing command line arguments or parameters to program when executing
- When main is called, it is called with two arguments: first is `argc` for arguments count, ans second is `argv` for argument value.
- Hence *main(int argc, char *argv[])*.....
- By convention, *argv*[0] is name by which program is invoked, so, *argc* is at least 1. When it is 1, there are no command line arguments.
- So, for, $ prog1 echo hello world $<$ *enter* $>$

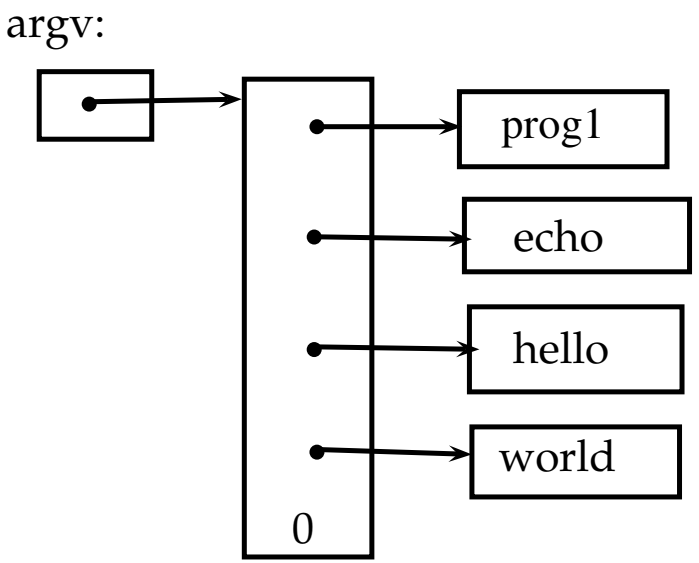


Figure 4: Pointers and arrays

Listing Command Line arguments

```
#include <stdio.h>
/* echo command-line arguments cmd.c;*/
int main(int argc, char *argv[])
{
    int i;
    for (i=0; i < argc; i++)
        printf("%s ", argv[i]);
    printf("\n");
    return 0;
}
```

Implementing Linux Command(cat): $ cat2 filename

```
/* cat2.c */
#include <stdio.h>
int main(int argc, char *argv[])
{
char ch;
FILE *pi;
pi = fopen(argv[1], "r");
while( (ch=getc(pi)) != EOF)
  putc(ch, stdout);
fclose(pi);
return 0;
}
```

Implementing Linux Command (cp): $ cp2 filename1 filename2

```
/* cp2.c */
#include <stdio.h>
int main(int argc, char *argv[])
{
char ch;
FILE *pi, *po;
pi = fopen(argv[1], "r");
po = fopen(argv[2], "w");
while( (ch=getc(pi)) != EOF)
  putc(ch, po);
fclose(pi);
fclose(po);
return 0;
}
```

How to check if copied properly?

Listing a file's content

```
/* cat2.c */
#include <stdio.h>
int main(int argc, char *argv[])
{
char ch;
FILE *pi;
pi = fopen(argv[1], "r");
while( (ch=getc(pi)) != EOF)
  putc(ch, stdout);
fclose(pi);
return 0;
}
```

This program can be used for cat command, "$ ./a.out file1"
Alternatively, $ gcc -o cat2 cat2.c

Process control commands

- **ps:** to display status of process, in you terminal or in entire system
- **kill:** terminal a process
- **sleep:** to put a process in sleep mode
- **$ cmd &:** to run a process in background mode

```
/*infy.c*/
int main()
{
while(1);
return 0;
}
```

Process control commands

```
/* infy1.c */
int main(){
   while(1){
    sleep(4);
    puts("infy1 is live");
   }
return 0;
}
/* infy2.c */
int main(){
  while(1){
   sleep(2);
   puts("infy2 is live");
  }
return 0;
}
```

You can run both program as multi-processes by: "$ prog1 &" and "$ prog2 &". Later a program can be killed by "$ kill pidno", where pid is process id of a program.

C Functions

Types of C functions:

- Library Functions
- User defined functions

Library Functions: main(), printf(), scanf(), ..

User defined functions:

```
 #include <stdio.h>
 \\define function
void function_name(){
.......
}
int main(){
............
function_name();
............
}
```

Advantages of user defined : decompose the large program into small segs, execute when needed, large project can divided.

User defined function types

- Function with no arguments and no return value
- Function with no arguments and return value
- Function with arguments but no return value
- Function with arguments and return value.

Recursion

```
/* recursion.c */
#include <stdio.h>
int sum(int n);
int main(){
    int num,add;
    printf("Enter a positive integer:\n");
    scanf("%d",&num);
    add=sum(num);
    printf("sum=%d",add);
    return 0;
}
int sum(int n){
    if(n==0)
       return n;
    else
       return n+sum(n-1);    /*self call  to function sum() */
}
```

Function call: parameter passing by value

```
/* swap.c */
#include <stdio.h>
void swap(int a, int b);
int main(){
    int a, b;
    printf("Enter two numbers:\n");
    scanf("%d %d", &a, &b);
    swap(a, b);
    printf("The numbers are %d and %d\n", a, b);
return 0;
    }
void swap(int a, int b){
     int t;
     t = a;
     a = b;
     b = t;
}
```

parameter passing by reference

```
/* swap2.c */
#include <stdio.h>
void swap(int *a, int *b);
int main(){
    int a, b;
    printf("Enter two numbers:\n");
    scanf("%d %d", &a, &b);
    swap( &a, &b);
    printf("The numbers are %d and %d\n", a, b);
return 0;
    }
void swap(int *a, int *b){
     int t;
     t = *a;
     *a = *b;
     *b = t;
}
```

Passing value v/s reference

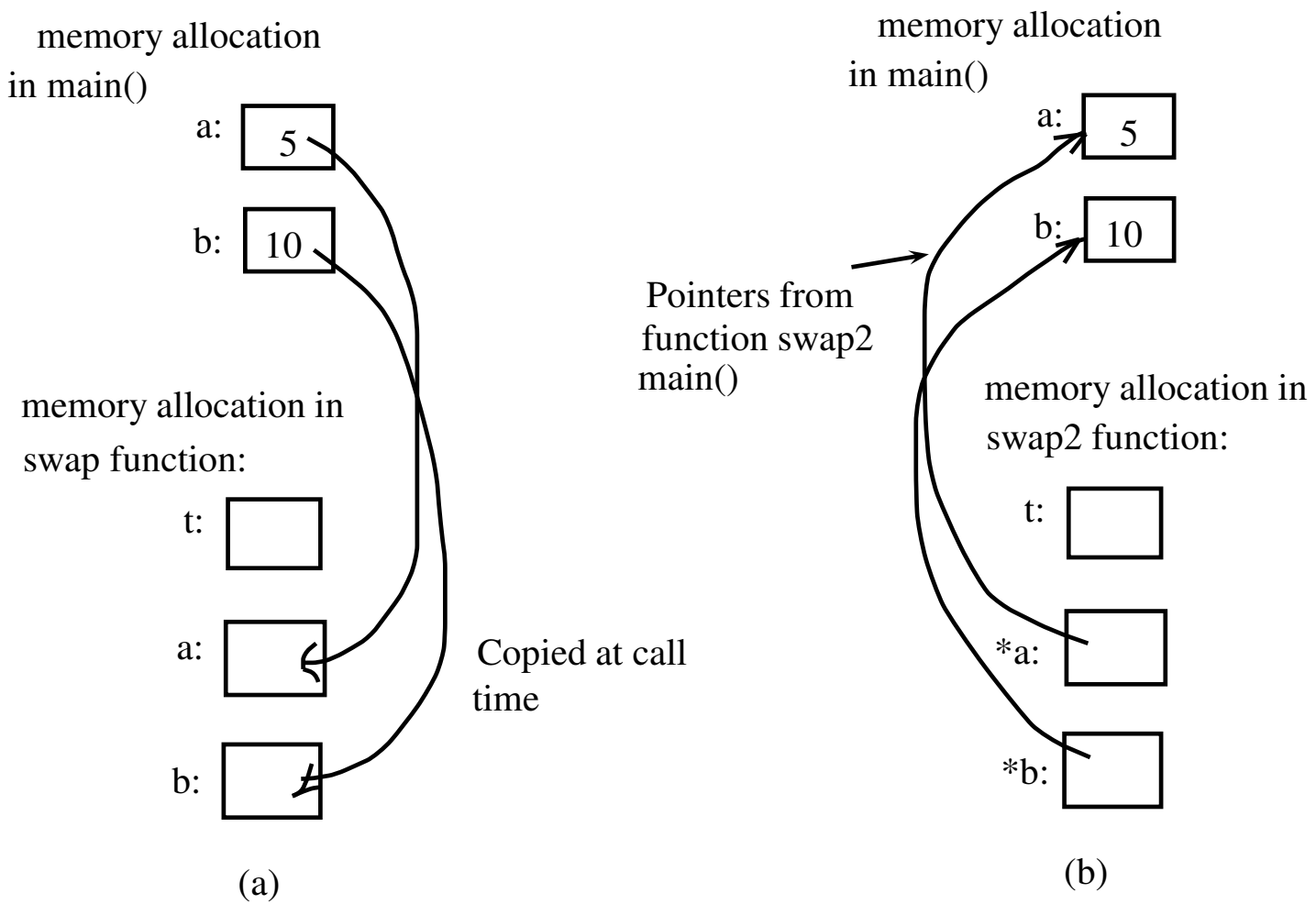


Figure 5: Passing (a) Value (b) Reference.

Pointer constants

```
/* ptrconst.c*/
 #include<stdio.h>
int main(void)
{
    char ch = 'c';
    char c = 'a';
    char *const ptr = &ch; // A constant pointer
    ptr = &c; /* Trying to assign new address
       to a constant pointer. WRONG!!!!*/
    return 0;
}
```

Pointers to functions

- Just like pointer to characters, integers etc, we can have pointers to functions.
- int (*fptr)(int, int)
- A function pointer 'fptr' that can point to a function whose return type is 'int' and takes two integers as arguments.

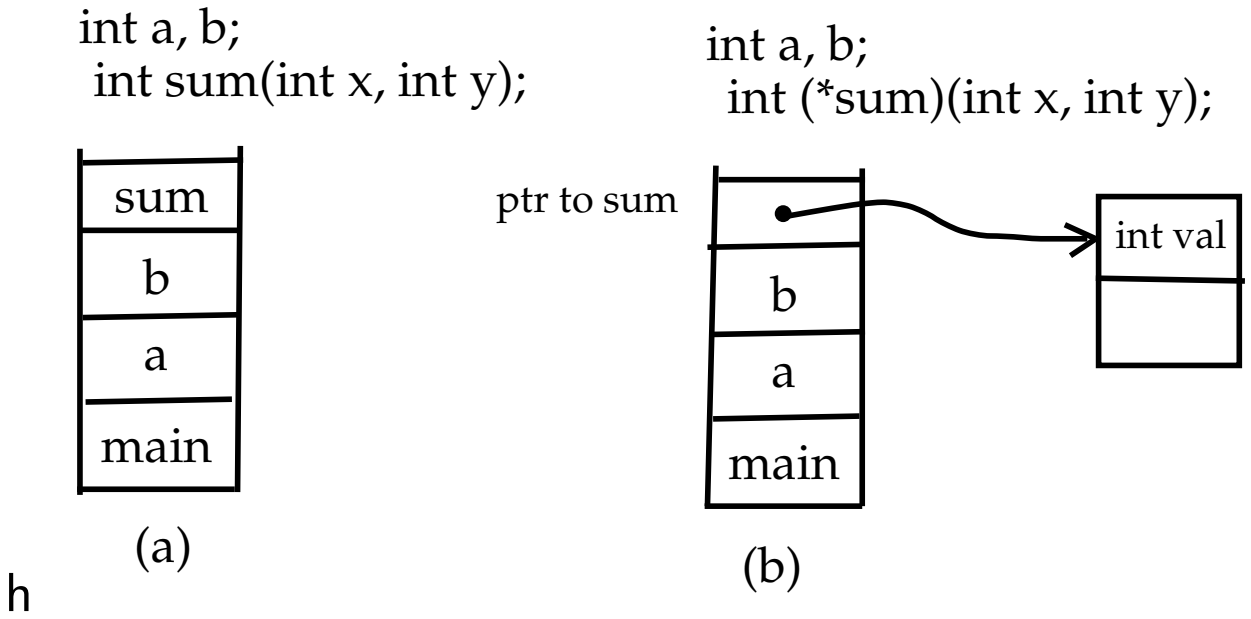
```
/* ptrfn.c*/
#include<stdio.h>
int func (int a, int b)
{
    printf("\n a = %d\n",a);
    printf("\n b = %d\n",b);
    return 0;
}
```

Pointers to functions

```
int main(void)
{
    int(*fptr)(int,int); // Function pointer
    fptr = func; // Assign address to function pointer
    func(2,3); // call function as ordinary
    fptr(20,30);  // call same through pointer
    return 0;
}
```

Name of the function can be treated as starting address of the function so we can assign the address of function to function pointer using function's name(above).

Pointers to functions



h

Figure 6: (a) Normal function (b) Pointer function.

Pointers to functions

```
int addInt(int n, int m) {
    return n+m;
}

int (*functionPtr)(int,int);

functionPtr = &addInt;
```

Now that we have a pointer to the function, lets use it:

```
int sum = (*functionPtr)(2, 3); // sum == 5
```

Passing the pointer to another function as argument:

```
int add2to3(int (*functionPtr)(int, int)) {
    return (*functionPtr)(2, 3);
}
```

Pointers to functions

```
int addInt(int n, int m) {
    return n+m;
}

int (*functionPtr)(int,int);

functionPtr = &addInt;
```

Now that we have a pointer to the function, lets use it:

```
int sum = (*functionPtr)(2, 3); // sum == 5
```

Passing the pointer to another function as argument:

```
int add2to3(int (*functionPtr)(int, int)) {
    return (*functionPtr)(2, 3);
}
```