Warning: These notes are not complete, it is a Skelton that will be modified/add-to in the class. If you want to us them for studying, either attend the class or get the completed notes from someone who did

CSE2301

Arrays and Pointers

These slides are based on slides by Prof. Wolfgang Stuerzlinger at York University

Arrays

- Data structure
- Grouping of data **of the same type**
- Indicated with brackets containing positive integer constant or expression following identifier
  - Subscript or index
- Loops commonly used for manipulation
- Programmer sets size of array explicitly

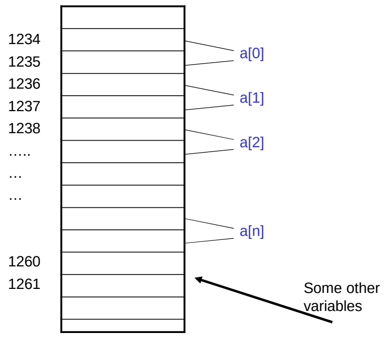
Arrays

- Syntax
  - ***type name*[*value*];**
- Example
  - **Int bigArray[10];**
  - **Double a[3];**
  - **Char grade[10], oneGrade;**

Arrays

- Declare the array → allocates memory
  int score[5];
  - Declares array of 5 integers named "score"
  - Similar to declaring five variables:
    int score[0], score[1], score[2], score[3], score[4]
- Individual parts called many things:
  - Indexed or subscripted variables
  - "Elements" of the array
  - Value in brackets called index or subscript
    - Numbered from 0 to size - 1

Arrays



Initialization

- In declarations enclosed in curly braces

```
int a[5] = {11,22};
```

Declares array a and initializes first two elements and all remaining set to zero

```
int b[ ] = {1,2,8,9,5};
```

Declares array b and initializes all elements and sets the length of the array to 5

Array Access

- X=ar[2];
- ar[3]=2.7;
- What is the differenc ebetween ar[i]++, ar[i++], ar[++i];

Strings

- No `string` type in C
- Char gretings[]=“hello”

H	e	l	l	o	\n

Pointers

- Memory address of a variable
- Declared with data type, * and identifier
  `type * pointer_var1, * pointer_var2, …`
- Example.
  double * p
  int *p1, *p2;
- There has to be a * before EACH of the pointer variables

- Use the **"address of"** operator (&)
- General form:

pointer_variable = &ordinary_variable

Name of the pointer

Name of ordinary variable

Using a Pointer Variable

- Can be used to access a value
- Unary operator * used
  - * pointer_variable
  - In executable statement, indicates value
- Example

```
int *p1, v1;
v1 = 0;
p1 = &v1;
*p1 = 42;
printf(“%d\n“,v1);
printf(“%d\n,*p1);
```

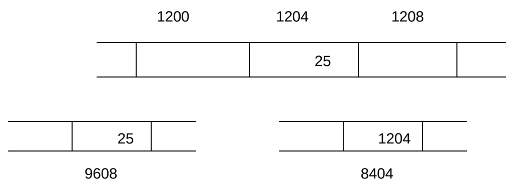
Output:
```
42
42
```

Pointer Variables

```
x = 25;
y = x;
z = &x;
```

int x,y;

int * z;

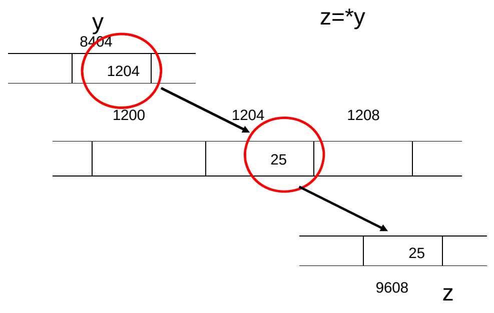


Pointer variables

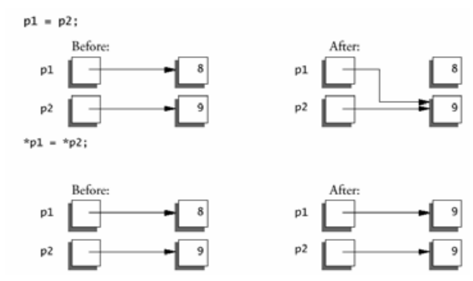
~~z= 0x12345A~~ BAD idea

Instead, use z=& another-vriable

Pointer Types



Pointers


```
p1 = p2;
```
```
*p1 = *p2;
```

Pointers

- identifier of an array is equivalent to the address of its first element
  - int numbers [20];
    int * p;

    p = numbers   // Valid
    numbers = p   // Invalid
- **p** and **numbers** are equivalent and they have the same properties
- Only difference is that we could assign another value to the pointer **p** whereas **numbers** will always point to the first of the 20 integer numbers of type int

Pointer Arithmetic

- int *x, *y
- int z;
- Can do
  - z=x-y;
  - x=NULL;
  - if(c==NULL)
  - Also, what is void * ?

Pointer Arithmetic

- int x[10];
- what is x[i] is it the same as *(x+i)
- What is the unit of x++ or x+5 5 what?
- Two functions
- void swap(int x, int y)
- void swap(int *x, int *y)

Pointers

- void * (pointer to a void) is the generic pointer replacing char *)
- Legal: add/sub a pointer and an integer, subtracting and comparing 2 pointers to members of the same array, and assigning or comparing to zero.
- Illegal add, multiply or divide 2 pointers, or assign one type to another type except void * without a cast.
- Any pointer can be cast to void * and back again without loss of information (used for pointer argument).

Functions

- Arrays passed to a functions are passed by reference.
- The name of the array is a pointer to its first element
- `strcpy(char dest[], char src[]);`
- Note that does not copy the array in the function call, just a *reference* to it.

String Functions

- Man the following functions
  - **`strcpy`**
  - **`strcmp`**
  - **`strcat`**
  - **`trlen`**
  - **`strchr`**
  - **`strstr`**

Multi-Dimensional Arrays

```
Int a[3][3];
```

```
Int a[3][3] = {
 {1,2,3},
 {4,5,6},
 {7,8,9}};
```

```
Int a[ ][3] = {
 {1,2,3},
 {4,5,6},
 {7,8,9}};
```

~~Int a[ ][ ] = {~~
~~{1,2,3},~~
~~{4,5,6},~~
~~{7,8,9}};~~

Multi-Dimensional Arrays

- Multi-dimensional arrays are array of arrys
- For the previous example, m[0] is a pointer to the first row.
- Lay out in memory

M[0][0]	M[0][1]	M[0][2]	M[1][0]	

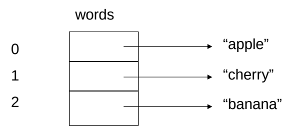
Multidimensional arrays

```
• #include <stdio.h>
• int main() {
• float *pf;
• float m[][3]={       {0.1, 0.2, 0.3},
•                      {0.4, 0.5, 0.6},
•                      {0.7, 0.8, 0.9} };
• printf("%d  \n",sizeof(m));
•  pf=m[1];
• printf("%f   %f   %f  \n",*pf, *(pf+1), *(pf+2));
• printf("%f   %f   %f  \n",*pf, *(pf++), *(pf++));
• }
```

```
36
0.4000   0.5000   0.6000
0.6000   0.5000   0.4000
```

Array of Pointers

- Char *words[]={“apple”, “cherry”, “banana”};
- Words is an array of pointers to a char, each element of words words[0], … is a pointer to a char.



Pointers to Pointers

- Pointers can point to integers, floats, chars, and other pointers.

```
int **j;
int *i;
int k=10;
i=&k;
j=&i;
printf(“%d   %d   %d\n”,j,i,k);
printf(“%d   %d   %d\n”,j,*j,**j);
printf(“%x   %x   %x\n”,j,*j,**j);
```

On my system

```
-1073744352  -1073744356  10
-1073744352  -1073744356  10
bffff620     bffff61c     a
```

Arrays vs. Pointers

- What is the difference between the last example and
- `char words[][10] = { “apple”,`
- `“cherry”,`
- `“banana”};`

strcpy

```
void strcpy(char *s, char *t) {
 int i;
 i=0;
 while( (s[i] = t[i]) != ‘\0’ )
        i++;
}
```

strcpy

```
void strcpy(char *s, char *t) {
 while( (*s = *t) != ‘\0’ ) {
        s++;
        t++;
        }
}
```

strcpy

```
void strcpy(char *s, char *t) {
 while( (*s++ = *t++) != ‘\0’ ){};
}
```

EX

```
char *words[] = { “apple”,
                  “cherry”,
                  “banana”};
                       +1
Char **p;              a
 p=words;              e
 printf("%c\n", **p);  o
 printf("%c\n",*(*(p+1)+2));
 printf("%c\n",*(*(p+2)+2)+1);
```

Pointers to Whole Arrays

```
Char (*p2)[100];
 char name[100];
 char *p1;

 p1=name;
 p2=name; // What is the difference?
      Consider p1+1 and p2+1
```

Command-Line Arguments

- Up to now, we defines main as `main()`
- Usually it is defined as
- `main(int argc, char*argv[])`
- argc is the number of arguments
- argv is a pointer to the array containing the arguments.
- argv[0] is a pointer to a string with the program name

Command-Line Arguments

```
main(int argc, char *argv[]) {
int i;
printf("Number of arg=%d\n",argc);
for(i=0;i<argc,i++)
        printf(%s\n",argv[i]);
}
```

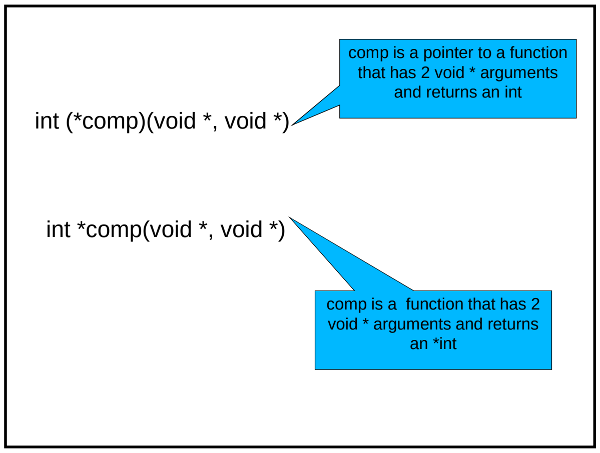
```
a.out
Number of arg=1
a.out
```

```
a.out hi by 3
Number of arg=4
a.out
hi
by
3
```

What if ./a.out

Pointers to Functions

- Although functions are not variables, it is posible to assign a pointer to a function.
- That pointer could be manipulated, assigned, placed on arrays, or passed/returned to/by functions.



int (*comp)(void *, void *)

comp is a pointer to a function that has 2 void * arguments and returns an int

int *comp(void *, void *)

comp is a function that has 2 void * arguments and returns an *int

Example

```
main() {
 float y,z;
float (*myfun)(float);
x=0.76;
y=acosf(x);
printf("%f\n",y);
printf("==========\n");
myfun=acosf;
z=myfun(x);
printf("%f\n",z);
}
```

```
0.707483
==========
0.707843
```

Complicated declaration

- int *f( );
  - f returns a pointer to int
- int (*pf)()
  - pf is a pointer to a function that returns int
- char **argv
  - argv is a pointer to pointer to char
- int (*daytab)[13]
  - daytab pointer to an array [13] of int

Complicated Declaration

- char (*(*x())[]) ( )
  - x is a function returning pointer to array of pointers to function returning char
- char (*(*x[3]) ( )) [5]
  - x is an array[3] of pointer to function returning pointer to array[5] of char