Lecture 2

CIS 208

Wednesday, January 18th, 2005

Why declare at beginning?

- Original compilers were one-pass.
- No look-ahead.

Scanf

- In <stdio.h>
- Formatted Console Input
- ‘Plugs’ in input into memory.

Scanf

int scanf(const Char *control_string,…,);

Returns # of successfully assigned values

Control_string

- Much like printf
- format specifiers - page 500
- Must exactly match.

```
char a, b;
int j,k;
scanf("%c%c%d%d",&a,&b,&j,&k);
```

& operator

- ‘address of ‘ operator
  - &a read as ‘address of variable a’ or ‘memory location of variable a’
- represents a memory location
- value determined at runtime
- prints as a large number

scanf cont…

- Data plugged into memory location
- Waits until enough data in stream.
- Must give address
- OR ELSE…..

scanf : electric boogaloo

- white spaces are characters
  - includes \t, \n
- If user types in 1 character, then presses enter, there are actually 2 characters in the input stream.

Stream Buffering

- Left over input, waiting…
- Scanf will gather next time.
- Must clear out the buffer

Flushing out the stream

- Remove extra data in input stream.
- Best way:

```
while (getc(stdin) != '\n');
```

- gets all extra characters in stream, stops when it reads a new line (enter)

Scanf_example

Date Types

- 5 Fundamental Types

char, int, float, double, void

type	size	range
char	8	0 to 256
int	16	-32,767 to 32767
float	32	Six digits of precision
double	64	Ten digits of precision.

Modifiers

- signed, unsigned, long, short
- short depends on machine word size
- char is inherently unsigned
  - everything else is signed by default

ascii values

- characters have numerical value

A = 65, B = 66, C = 67, etc
a = 97, b = 98, c = 99, etc

See the pattern/advantage?

Ascii Arithmetic

- Addition and Subtraction apply to chars.

```
char xy = 'B';
xy = xy + 32;
printf("%c\n",xy);
xy = xy –1;
printf("%c\n",xy);
```

Explicit Casting

- Switch one fundy type into another

```
int a;                int a;
float b;              char b;
b = (float) a;        a = (int) b;
```

- Data and precision can be lost.

Variable declarations

- 3 places
  - Inside functions: local variables
  - in def. of function param.: Formal params.
  - outside all functions: global variable

what knows what?

- local: only known to code segment that declares it
- param: only known to function that declares it
- global: known to everyone, even other files

Local examination

- local variables declared at start of code segment
- code segments defined by { }
- local vars. not known outside of segment

code segments

```
void f(void) {
int I;
I = 10;
int j;
j = 20;
}
```

```
void f(void){
int I;
I = 10;
{int j;
  j = 20;}
}
```

```
void f(void){
int I;
I = 10;
{int j;
j = 20;}
I = j;
}
```

Variable types cont.

- const : constant variable. Can't change,ever
- extern: global variable declared in another file
- register : puts value in a register. Why?

static

- static label further alters variables
  - only known to declared block
- static local : value is known to local only
  - remains in persistent memory
  - values bridge function calls

static example

```
void foo(int i) {
static int a;
if (i ==0) a = 0;
else --a;
return a;
}
```

static global

- global to all functions in file
- Not known to other files.

Control flow

- Most important: No Booleans
- 0 is false and non-zero is true.

flow constructs

- if – else
- while
  - do –while
- for

Same as java

if - else

```
if (expression)
{ commands}  //if expression != 0
else { }
```

- As in java, single commands don’t need brackets

```
int (j < k)
{j = 1;}
else {j = 2;}
```

```
int (j < k)
 j = 1;
else j = 2;
```

for loops

- same as java
- may have multiple parts in each section
  - use comma as separator
- some sections can be skipped

for loops cont.

```
for (j = 1,k =2; j +k < 10; ++k, ++j)
```

section 2 can't have multiple parts, use || or &&

```
for(;j < k;)
```

more loop stuff

- infinite loop

  for(;;)

- break; stops current loop and fails test

- continue; stops current iteration of loop.

assignment 1