ESC101N
Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

Type conversion rules

- For relational operators, lower type is *promoted* to higher type
- Highest type is `double`, followed by `float`, then `int` and finally the lowest type is `char`

Type conversion rules

- For relational operators, lower type is *promoted* to higher type
- Highest type is `double`, followed by `float`, then `int` and finally the lowest type is `char`
- For assignments, value of right side is converted to type of left

Type conversion rules

- For relational operators, lower type is *promoted* to higher type
- Highest type is `double`, followed by `float`, then `int` and finally the lowest type is `char`
- For assignments, value of right side is converted to type of left
- Code snippets like

  ```
  double d = 12.56;
  int i = d;
  i = 34.56;
  ```

  works!
- Value of `i` is first `12`, and then `34`

Type conversion rules

- For relational operators, lower type is *promoted* to higher type
- Highest type is `double`, followed by `float`, then `int` and finally the lowest type is `char`
- For assignments, value of right side is converted to type of left
- Code snippets like

  ```
  double d = 12.56;
  int i = d;
  i = 34.56;
  ```

  works!
- Value of `i` is first 12, and then 34
- Explicit type casting can be done by

  ```
  double d = -12.56;
  int i = (int)d;
  ```

- Value of `i` gets *truncated* to −12

Type conversion rules

- For relational operators, lower type is *promoted* to higher type
- Highest type is `double`, followed by `float`, then `int` and finally the lowest type is `char`
- For assignments, value of right side is converted to type of left
- Code snippets like

  ```
  double d = 12.56;
  int i = d;
  i = 34.56;
  ```

  works!
- Value of `i` is first 12, and then 34
- Explicit type casting can be done by

  ```
  double d = -12.56;
  int i = (int)d;
  ```

- Value of `i` gets *truncated* to −12
- Conversion from higher type to lower type may lose information
- Conversion from lower type to higher type should not lose information

Type conversion rules

- For relational operators, lower type is *promoted* to higher type
- Highest type is `double`, followed by `float`, then `int` and finally the lowest type is `char`
- For assignments, value of right side is converted to type of left
- Code snippets like

  ```
  double d = 12.56;
  int i = d;
  i = 34.56;
  ```

  works!
- Value of `i` is first 12, and then 34
- Explicit type casting can be done by

  ```
  double d = -12.56;
  int i = (int)d;
  ```

- Value of `i` gets *truncated* to −12
- Conversion from higher type to lower type may lose information
- Conversion from lower type to higher type should not lose information
- Avoid

Inputting multiple characters: version 1

- scanf(‘‘%c’’, &c) reads a single character
- If multiple characters are need to be read

```
#include <stdio.h>
int main()
{
    char a,b,c;

    printf("Enter first character\n");
    scanf("%c", &a);

    printf("Enter second character\n");
    scanf("%c", &b);

    printf("Enter third character\n");
    scanf("%c", &c);

    printf("%c\n%c\n%c\n", a, b, c);
}
```

Inputting multiple characters: version 1

- scanf(‘‘%c’’, &c) reads a single character
- If multiple characters are need to be read

```
#include <stdio.h>
int main()
{
    char a,b,c;

    printf("Enter first character\n");
    scanf("%c", &a);

    printf("Enter second character\n");
    scanf("%c", &b);

    printf("Enter third character\n");
    scanf("%c", &c);

    printf("%c\n%c\n%c\n", a, b, c);
}
```

- Does not work!
  - “Enter” is read as a character

Inputting multiple characters: version 2

- “Enter” is not printed
- Can be printed as an integer

```
#include <stdio.h>
int main()
{
    char a,b,c;

    printf("Enter first character\n");
    scanf("%c", &a);

    printf("Enter second character\n");
    scanf("%c", &b);

    printf("Enter third character\n");
    scanf("%c", &c);

    printf("%d\n%d\n%d\n", a, b, c);
}
```

- Note the automatic type conversion

Inputting multiple characters: version 3

- Read all of them at one go, and press “Enter” only at the end

```
#include <stdio.h>
int main()
{
    char a, b, c;

    printf("Enter three characters\n");
    scanf("%c", &a);
    scanf("%c", &b);
    scanf("%c", &c);

    printf("%c\n%c\n%c\n", a, b, c);
    printf("%d\n%d\n%d\n", a, b, c);
}
```

Inputting multiple characters: version 3

- Read all of them at one go, and press “Enter” only at the end

```
#include <stdio.h>
int main()
{
    char a, b, c;

    printf("Enter three characters\n");
    scanf("%c", &a);
    scanf("%c", &b);
    scanf("%c", &c);

    printf("%c\n%c\n%c\n", a, b, c);
    printf("%d\n%d\n%d\n", a, b, c);
}
```

- What if the number of characters to be read is variable?
- A loop needs to be used

Inputting variable number of characters

- Read all of them at one go, and press “Enter” only at the end

```
#include <stdio.h>
int main()
{
    int i, n;
    char c;

    printf("Enter the number of characters\n");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
    {
        scanf("%c", &c);
        printf("%c\n", c);
        printf("%d\n", c);
    }
}
```

Inputting variable number of characters

- Read all of them at one go, and press "Enter" only at the end

```
#include <stdio.h>
int main()
{
    int i, n;
    char c;

    printf("Enter the number of characters\n");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
    {
        scanf("%c", &c);
        printf("%c\n", c);
        printf("%d\n", c);
    }
}
```

- scanf requires "Enter" before it can read
- Typed characters are remembered as input

Using getchar()

- getchar() reads a single character

```
#include <stdio.h>
int main()
{
    int i = 0;
    char c;

    while ((c = getchar()) != EOF)  // Stop input with
        Ctrl-D
    {
        printf("%c\n", c);
        i++;
    }

    printf("Number of characters input is %d\n", i);
}
```

Using getchar()

- getchar() reads a single character

```
#include <stdio.h>
int main()
{
    int i = 0;
    char c;

    while ((c = getchar()) != EOF)  // Stop input with
        Ctrl-D
    {
        printf("%c\n", c);
        i++;
    }

    printf("Number of characters input is %d\n", i);
}
```

- Number of characters include “Enter”
- EOF is a special value to indicate end of input

printf special characters

Character	Interpretation
\a	Bell
\b	Backspace
\n	New line
\t	Tab
\0	Null character
\'	Single quote
\''	Double quote
\\	Backslash

- putchar(c) prints the character c

```
char c = '\t';
putchar(c);
```